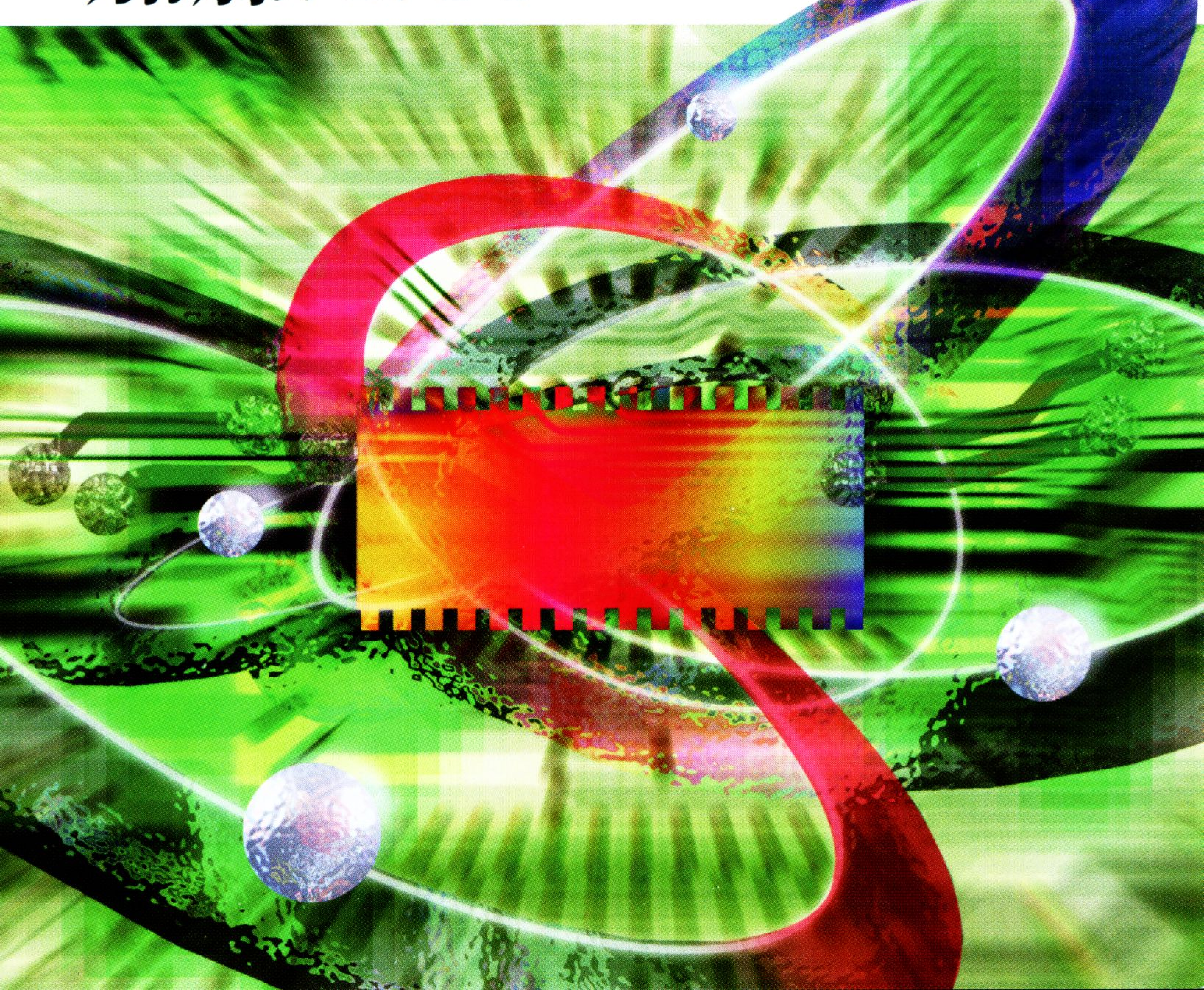
BLUE BACKS

動かしながら理解する CPUの仕組み

パソコンの中心はどうなっているのか

CD-ROM付

加藤ただし



かとう・ただし（加藤　肇）

一九五八年、奈良県吉野郡に生まれる。現在の職業能力開発総合大学校を卒業後、日本電産（株）に入社。以後十年間、電子回路の設計エンジニアとして従事。一九九一年より教職に就く。高度ポリテクセンターやポリテクカレッジ新潟などを経て、現在ポリテクカレッジ秋田で学務援助課長を務める。学生の入学から就職に至る学務業務全般に加え、在職者訓練・離職者訓練などの支援業務、若手教員の育成にあたっている。著書に『図解・わかる電子回路』（共著）『CD-ROM付　電子回路シミュレータ入門　増補版』『図解 つくる電子回路』（以上、講談社ブルーバックス）など。

動かしながら理解するCPUの仕組み CD-ROM付

パソコンの中心はどうなっているのか

加藤ただし　著

ブルーバックス

■付属アプリケーションが動作するパソコンOS

Windows 7, Windows Vista, Windows XP

装幀／芦澤泰偉・児崎雅淑
カバーイラスト・もくじ／中山康子

はじめに

初めてこの世に登場したコンピュータは、真空管を使った巨大なもので、それは家庭にはまったく縁のないものでした。やがて、トランジスタや集積回路（略してIC）が発明されて、小型化が急速に進みました。その結果、現在多くの方々が、ノートパソコンを日常的に使い、ビジネスの現場から家庭にまで広く普及しているのです。

しかし、パソコン以上に成長を続けているのは、むしろ「組み込みシステム」です。外から見たとき、誰もがコンピュータであると理解できるパソコンとは異なり、見えないところで活躍しています。

組み込みシステムは、それまで数多くの電子部品で構成されてきた電子回路に、目的を限定した小さなコンピュータ、「マイコン」を組み込んだものです。単に、複雑な電子回路が１個のマイコンに置き換わったというだけではなく、電子回路では実現するのがとても面倒なことでも、マイコンにプログラムすることによって難なくできるようになってしまったのです。そのため、非常に小型で高性能な電子機器が作れるようになりました。

携帯電話が便利に改良されていくのも、家電製品に全自動が増えたのも、すべてマイコンのおかげです。

つまり、現在の「コンピュータ」は、おおむねパソコンと組み込みシステムを総称したものであると言えます。

1　本書の目的

さて、コンピュータ化が進んで、生活が便利になるのは歓迎すべきところですが、一方メーカー側では、マイコンなど電子回路に近いコンピュータを扱えるエンジニアが、不足することになりました。

これを解決するには、若い学生の皆さんに「コンピュータの仕組みを知りたい」という願望を持っていただかなくてはなりません。

ところが、電子回路の知識なしではコンピュータの仕組みは理解できません。ですから、必ずディジタル電子回路の基礎から学びます。実際、教壇に立つ筆者もそのように教えています。

困ったことに、コンピュータは電子回路でありながら、パソコンの形になっていると、電子回路を知らなくても、市販された完成品のプログラムを使って仕事をしたり、遊ぶことができます。

これでは、堅苦しい電子回路の世界に足を踏み入れたくなくなるのも無理はありません。結果として、パソコンとしてのコンピュータにだけ触れることで、十分な満足感を得てしまうのです。

そこで、電子回路としてのコンピュータを初めて学ぶ方々のために、数字の変化だけを観察しながら、コンピュータに親しんでいただく本をつくることにしました。

2　Z80を選んだわけ

コンピュータは、大規模な電子回路システムです。その中枢部が、CPU（Central Processing Unit：中央演算処理装置）です。

CPUにもさまざまな形態があり、先に述べたように、最近はマイコンと呼ばれるCPUが産業界での主流です。しかし、基本的な構成や仕組みは、真空管の時代からほとんど変わっていません。

こうした背景を考慮した末に、かつて米ザイログ社が開発した８ビットマイコン「Z80」を選びました。パソコンの発展に貢献したCPUの１つです。

Z80が開発されたのは1976年ですから、言うまでもなく昔のマイコンであり、古典的なイメージがあるのはやむをえません。しかし、今でも筆者は、教室でZ80を教えています。しかも、いまだにZ80のファンです。

卒業制作でも、10年以上にわたって、このマイコンをコア（中枢部）にしたLSI（大規模集積回路）を、作品のコントローラとして組み込んでいます。

長年Z80を授業で取り上げてきた最大の理由は、CPUの基本的な仕組みを短期間で理解することができ、プログラムの構造もわかりやすいという点です。教育的な価値が突出していると考えているのです。

Z80をCPUに構成したマイコンシステムは、ひじょうにシンプルで、構成要素は　①CPU　②メモリ　③I/Oポート（アイオーポート：周辺装置へのインターフェース）　のわずか３つです。

２つ目に、すべてのデータが８ビット（２進数で８桁）を単位にして管理されているという点です。

プログラムの各命令は、８ビット（１バイト）の整数倍（１～４バイト）でつくられています。また、メモリのアドレスバスは８ビット×２の16ビットです。８ビットという「メモリの基本単位」で設計されていることが重要です。

３つ目は、Z80が単一の製品だったという点です。

多くの教育機関では、教育用マイコンとして、米マイクロチップ・テクノロジー社のPIC（Peripheral Interface Controllerの略称）や、ルネサス テクノロジー社のH8という製品が選ばれています。

これら広く普及しているCPUは、たいてい初めから「シリーズ化」されており、その中から用途に応じて、最適なものを選ばなくてはなりません。

シリーズ化されたCPUでは、多くの場合、「コア」と呼ばれるCPU内部の中枢部の設計を固定して、周りに配置する機能回路の種類や数を変えることによって、多くの機種を生み出しています。

まったくの入門者にとって、学習対象のCPUは、迷わずじっくり取り組めることが重要です。

3　Z-Visionの役割

付録ソフトについて少し触れておきましょう。

本書の執筆にあたっては、システムロード社に協力をお願いしました。

「Z-Vision」は、本来Z80のプログラミングを効率的に行うためのデバッガ（24ページ参照）ですが、その中のシミュレータ部だけを抽出していただき、CD-ROMに収録しました。

ディジタル回路としてのシミュレータではなく、マイコンシステムの内部で、数値がどこからどこへ運ばれ、どこで演算されるのかをWindows画面上で観察するためのものです。

Z-Visionには、「電子回路」という分野の記号や用語は出てきません。また、使い方がシンプルですので、操作の習得に多くの時間を要しません。

慣れてくると、本文を読むことがなくなるでしょう。本書後半で紹介するZ80の命令から、ご自身で命令を組み合わせ、その動作をシミュレートしたくなるでしょう。

4　本書の構成

CPUは、それぞれ異なる機能を持った電子回路が、数多く集められたシステムです。そして、言うまでもなくその目的は、プログラムを実行してデータ（数値）の処理をすることです。

つまり、CPUの仕組みを理解する最短距離は、命令を実行する様子を直接観察することです。そのためには、どんなに短くても、命令を組み合わせたプログラムが必要です。

そこで、Z-Visionを併用して、次の手順で学習を進めます。

第1段階

シミュレータに表示されるのは、ほとんどが数字ですが、数字だけでは何をどのように処理しているのかがわかりません。

そこで半導体メーカは、あらかじめCPUが理解する「0」と「1」の数列（機械語）を、アルファベットの列に対応させています。このアルファベット列が、アセンブリ言語です。シミュレータには、機械語とアセンブリ言語を同時に表示する機能があるので、まずはまとまった仕事をするテストプログラムを作成します。

これによって、シミュレータの表示内容をもとに、CPUの内部構造を理解することができます。プログラミングをするのが目的ではありません。

第2段階

シミュレータにプログラムが導入されると、CPUを構成する各電子回路に数値が現れます。そして、命令を1つ実行するごとに、関連する電子回路上の数値が変化する様子を見ることができます。

命令を実行するときに、ひんぱんに動く電子回路もあれば、使用頻度の低い電子回路もあります。

CPUはひじょうに複雑なシステムですが、テストプログラムを動かすことによって、もっとも中心的に動作する電子回路がどこなのかを知ることができるのです。

第3段階

CPUの基本的な動きが見えるようになったら、演算結果を判断するときにもっとも重要な役割を果たす「フラグ」について理解を進めます。

そして、Z80CPUの代表的な命令を、個々に試していきます。ここでは、あえて機械語で短いテストプログラムを入力します。読者の皆さんの意識を、できるだけCPUに近づけるためです。

また、このとき画面には、アセンブリ言語も同時に表示されます。じっくりと1つ1つの命令を実行してみましょう。

筆者は、かつて企業生活で、突然マイコンを組み込むプロジェクトを任されたことがありましたが、学生時代に学んだZ80の知識がベースにありました。それゆえ、実務でも臆することなく取り組むことができました。

Z80で基本的なCPUの仕組みをきちんと理解していれば、おそらく、どのようなマイコンにでも挑戦することができるでしょう。

最後に、付録ソフトとして「Z-Vision付録版」を快く提供して下さったシステムロード社の青木勲氏と、執筆にあたってご助言いただいたブルーバックス出版部の皆さまに深くお礼申し上げます。

2009年11月末日　著者

もくじ

第7章 さまざまな命令語をためす 119

第１章　インストールの前に

本書には、CD-ROMが付属していますが、インストールする前に慣れていただきたいことがあります。それは「数字」の表現です。

CPUを初めて学ぶ方々が、いきなりZ-Visionのシミュレーション画面をご覧になると、面食らうかもしれません。

「はじめに」でも述べましたが、CPUの中はディジタル信号だけの世界です。これを数字で表すと「０」と「１」になり、CPU内部のあらゆる場所で、数字の０と１のわずか２種類の値が高速で変化するわけです。

これを２進数と言いますが、シミュレータでは、表示効率の高い16進数を主に使います。つまり、CPUを理解するには、「さまざまな数字の表現」と付き合っていかなくてはなりません。「０」と「１」だけではないのです。

そこでこの章では、シミュレータに表示される数字表現や、アセンブリ言語の表現の特徴に触れます。

1.1 コンピュータは2進数の世界

先にも述べたように、コンピュータもCPUもディジタル回路という電子回路の一種です。本来は、高い電圧（1）と低い電圧（0）だけで表記した2進数（Binary：バイナリ）でCPUの仕組みを理解すべきでしょう。

しかし、コンピュータという工業製品をつくったのは、10進数（Decimal：デシマル）で日常生活を送っている人間です。

つまり、人間が理解できる英単語をもとにつくられたプログラム言語の命令や、ものの数や金額を意味する数値を、「コンピュータという電子回路」の中に入れるために、やむを得ず2進数に変換しているのです。

2進数と10進数は、まったくの別モノです。

2進数には「2」という数字は存在せず、たとえば「1」という数値に「1」を足すと、すぐに桁（けた）が上がって、2桁の「10」になります。よく考えてみると10進数も、1つの桁に「10」を書くことはありません。

そのため、コンピュータに入門するときには、たいてい10進数から2進数に変換する方法や、逆に2進数で表記された数値の大きさを把握するために、10進数に変換する方法を練習します。

しかし本書では、こうした変換法について解説することを避けました。「2進数↔10進数」などの変換方法は、確かに基礎知識として重要ですが、それよりも「進数」ごとに目的が決まっていることの方がもっと重要です。

1.2　10進数は人間世界の数字

10進数は、私たち人間がふだん生活しているとき、物の数量を数えたり、金額を表したりするときに使います。

スーパーマーケットのレジ（キャッシュレジスタ）で、店員が2進数変換して、キーボードをたたいたりしませんし、100円のことをわざわざ「1100100円です」と口にすることは決してありません。

ただし、レジの内部では人知れず「1100100」という数字が、しっかりと存在しているのです。

あとで述べますが、プログラムの中で物の数を表現するときは、10進数のままでよいのです。

1.3　あえて2進数も使う

ところで、これからシミュレータを使ってCPUを学ぼうというのですから、人間にとってもっとも身近な10進数だけで扱えばよさそうなものです。しかし、コンピュータが電子回路である以上、どうしても2進数のままで扱わなくてはならない場面があります。

2進数は、電子回路の中の1本の信号線に、ある瞬間に現れている電圧について「高いか低いか」を表現したものです。

プログラムの中では、たとえばCPUにつながっている信号の出入り口、これをI/Oポート（アイオーポート）と呼びますが、信号線1本1本の電圧の状態を表すときに、2進数を使うのです。

わざわざ10進数に変換する必要はありませんし、もし

変換してプログラムに書き込んでしまうと、書き込んだ本人すら意味を思い出せなくて、ムダな時間を費やすことになるでしょう。

1.4 16進数はシミュレータに適する

10進数の世界で生活している私たち人間は、通常、桁数の多い2進数を読む機会がありません。

そのため、仮に2進数の数列と英単語が対応しているとしたときに、任意の2進数の数列を、即座に英単語と照らし合わせることは容易ではありません。

だからといって、無理やり10進数に変換した場合、前節で述べたように、電子回路としての信号線の意味が重要なときに、数値からそれを知るのに時間がかかります。

これらの問題点を、同時に解決する表記方法が16進数（Hexadecimal：ヘキサデシマル）です。

16進数は、10進数で0から15までの数値を、10個の数字（0～9）と、6個のアルファベット（A、B、C、D、E、F）で表したものです。

たとえば「A」は10進数の10のこと、「B」は11のことです。また、10進数で16まで数えた瞬間に、16進数では初めて桁上げし、「10」となります。

ここで重要なことは、2の4乗が16であるということです。つまり、2進数で4桁必要な数値も、16進数なら1桁で表せるのです。本来、CPUの中の信号の様子は、すべて2進数で表現すべきところですが、言うまでもなく2進数で表記すると、あまりにも桁が多くなります。

むしろ16進数は、プログラムを管理しやすくするための数値表現であると理解すべきでしょう。本書付録のシミュレータの画面では、特に設定を変えないかぎり、すべての数値が16進数で統一されています。

16進数は、限られた画面の面積の中に効率よくCPUの内部の状態を見渡すことにも、たいへん適しています。

私たちがプログラムのリストを作るとき、自分自身が理解しやすいように、２進数や10進数を使い分けて記載することはありますが、意識的に16進数を使う場面は限られています。それは、メモリの「アドレス（住所）」を記載するときです。

メモリのアドレスについては、4.2節で述べます。

10進数	2進数	16進数
1	1	1
10	1010	A
100	1100100	64
1000	1111101000	3E8
10000	10011100010000	2710
100000	11000011010100000	186A0
1000000	11110100001001000000	F4240
10000000	100110001001011010000000	989680

図1-1　効率よく表示できる16進数

1.5　CPUが実行する機械語

現代のコンピュータは、人間の世界から見ると、想像もつかないほど高速に大量のデータを処理できます。また、工業製品として、際限なく高速化と高機能化が進められてきました。しかし、CPUはあくまでディジタル回路であり、人間の世界を知りません。

「彼」はひたすら、規則的に送られてくるクロックのリズムに合わせて、目の前に置かれた「命令」を淡々と実行しています。

CPUが理解して実行する命令というのは、2進数の数列です。桁数を減らす16進数に置き換えたとしても、やはり数字であることに変わりはありません。これを「機械語」と呼びます。コンピュータが電子計算「機」だからです。

しかし、人間が機械語でプログラムするのはたいへんです。そこで、各機械語の命令の意味を、それに該当する英単語や略語で表記したものを使います。それが「アセンブリ言語」です。

企業のソフトウェア開発現場では、高級言語である「C言語」を主に使いますが、本書では、あえて機械語にもっとも近いアセンブリ言語だけを取り上げています。

詳しいことは後述しますが、Z80では、たとえば図1-2のような命令語を使います。これらは、米ザイログ社でデザインされた略語がもとになっています。具体例は、第7章をご覧ください。

命令の分類	命　令　語
ロード	LD
プッシュ、ポップ	PUSH、POP
エクスチェンジ	EX、EXX
比較	CP
算術演算	ADD、ADC、SUB、SBC、INC、DEC
論理演算	AND、XOR、OR
アキュムレータ操作	DAA、CPL、NEG
フラグ操作	CCF、SCF
ローテート	RLC、RRC、RL、RR、RLD、RRD
シフト	SLA、SRA、SRL
ビット操作	SET、RES
ジャンプ	JP、JR
コール、リターン	CALL、RET
入出力	IN、OUT

図1-2　Z80アセンブリ言語の主な命令語

第2章　シミュレータをインストールする

ここでは、「Z-Vision付録版」というシミュレータをインストールします。

Z-Visionは、埼玉県でマイコンシステムやソフトウェアの開発・販売を行っているシステムロード社のデバッガです。

デバッガ（debugger）とは、デバッグ（debug）つまり作成中のプログラムから誤りを見つけ、修正する作業を支援するソフトウェアのことです。

デバッガを使うと、仮想的にプログラムを動作させたり、特定のパラメータ（変数）を選んで一時的に変更したりすることができます。

また、パソコンとケーブル接続された、ターゲットボード（target board）と呼ばれる本物のCPUを搭載したプリント基板へプログラムを転送したり、ターゲットボード上のZ80CPU内部のパラメータを読み出して、パソコン画面上に表示させることができます。

ちなみに、Z-Vision付録版は、本書のために提供いただいた機能限定版のため、ターゲットボードとの通信機能は取り除かれています。

2.1　インストールの手順

以下は、Windows Vista Ultimateにインストールした様子です。掲載した図は、いずれも筆者のパソコン環境を画面キャプチャーしたものですので、フォルダの表示設定などによって読者ごとに多少異なると思います。

あくまで参考程度にご覧ください。

①付録のCD-ROMをディスクドライブに入れ、Windows上で、ディスクドライブのアイコンをダブルクリックして、開きます。「ZVW6Installer」フォルダが見えます。



図2-1　ディスクドライブを開く

②さらに、「ZVW6Installer」フォルダをダブルクリックすると、7つのアイコンが現れます。ここで、「setup」というファイルをダブルクリックします。



図2-2　setupプログラムを起動する

Windows Vistaでは、この場面で「ユーザアカウント制御」という警告ウィンドウが表示されますが、「→許可」のボタンをクリックして、先に進んでください。

③しばらくすると、図2-3の画面が現れます。「次へ>」をクリックします。

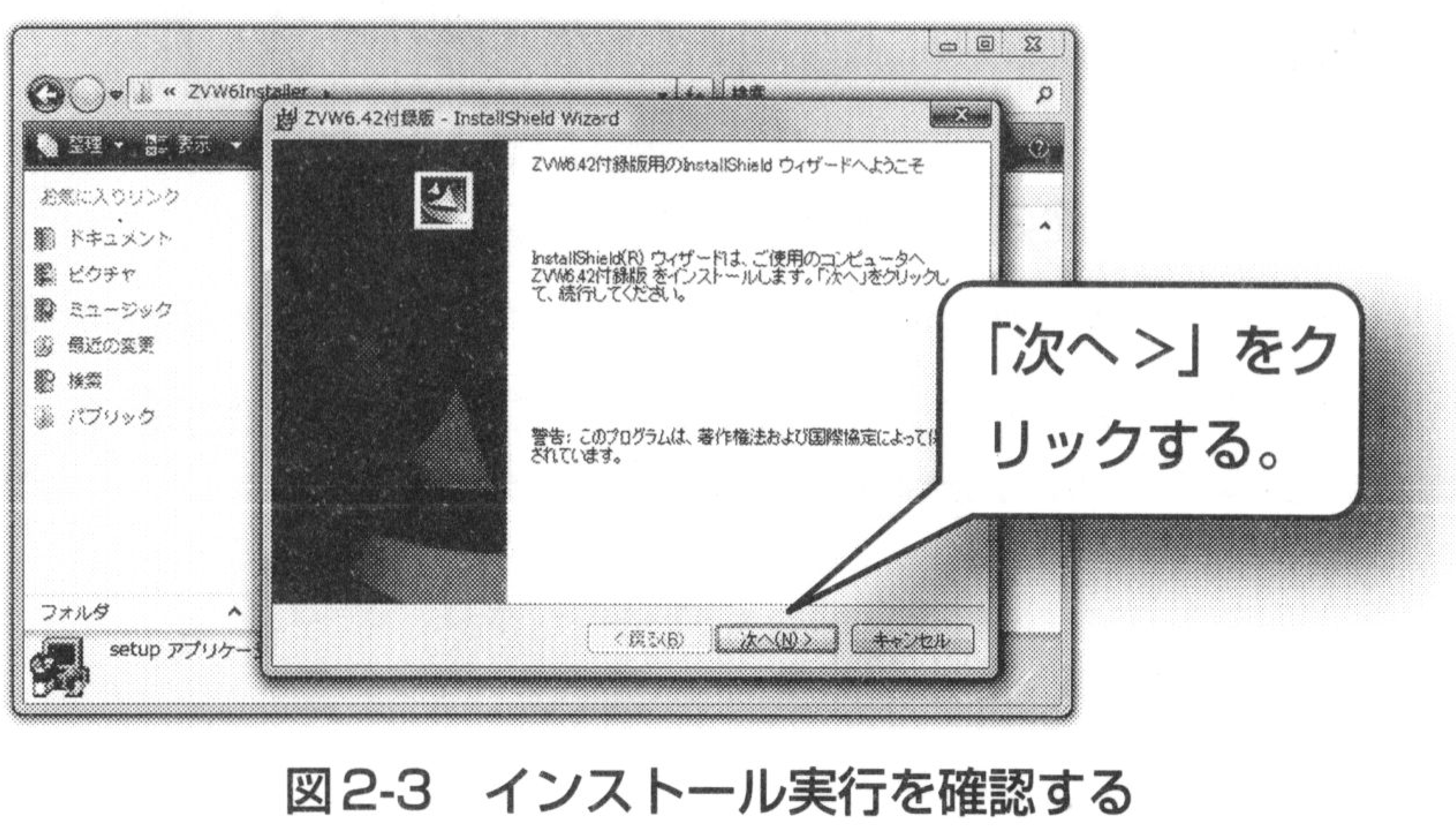


図2-3　インストール実行を確認する

④図2-4で「使用許諾契約の条項に同意します」の文字列の上でクリックし、左のラジオボタンが青く着色されるのを確認します。そして、ウィンドウ右下の「次へ>」をクリックします。

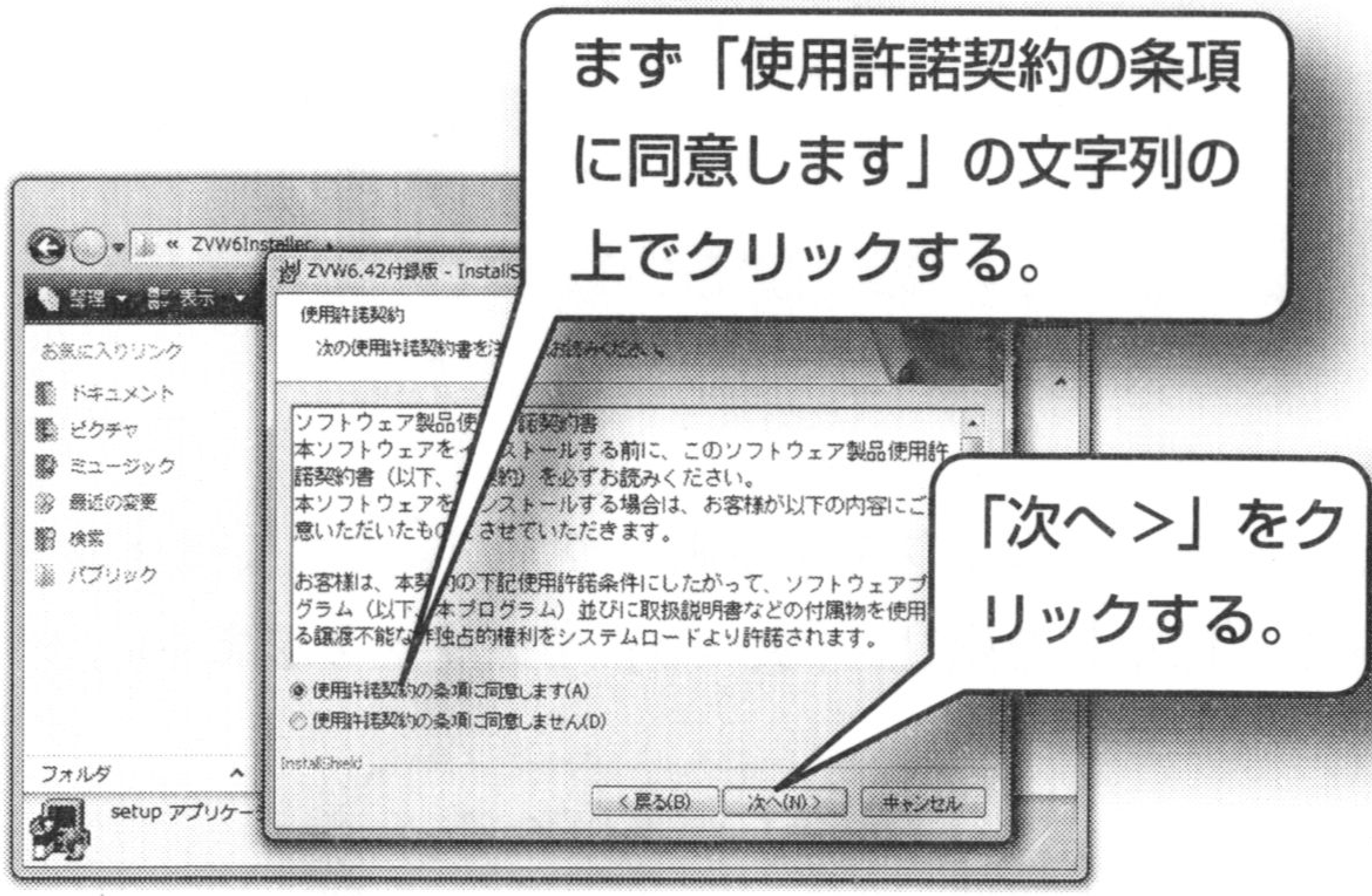


図2-4　使用許諾契約に同意する

⑤図2-5の画面では、読者の「ユーザ名」と「所属」を入力しますが、所属は特に必要ありません。

また、その下のラジオボタンについては、特に理由がない限り「このコンピュータを使用するすべてのユーザ」のままにしておきます。「次へ>」をクリックして次に進みます。

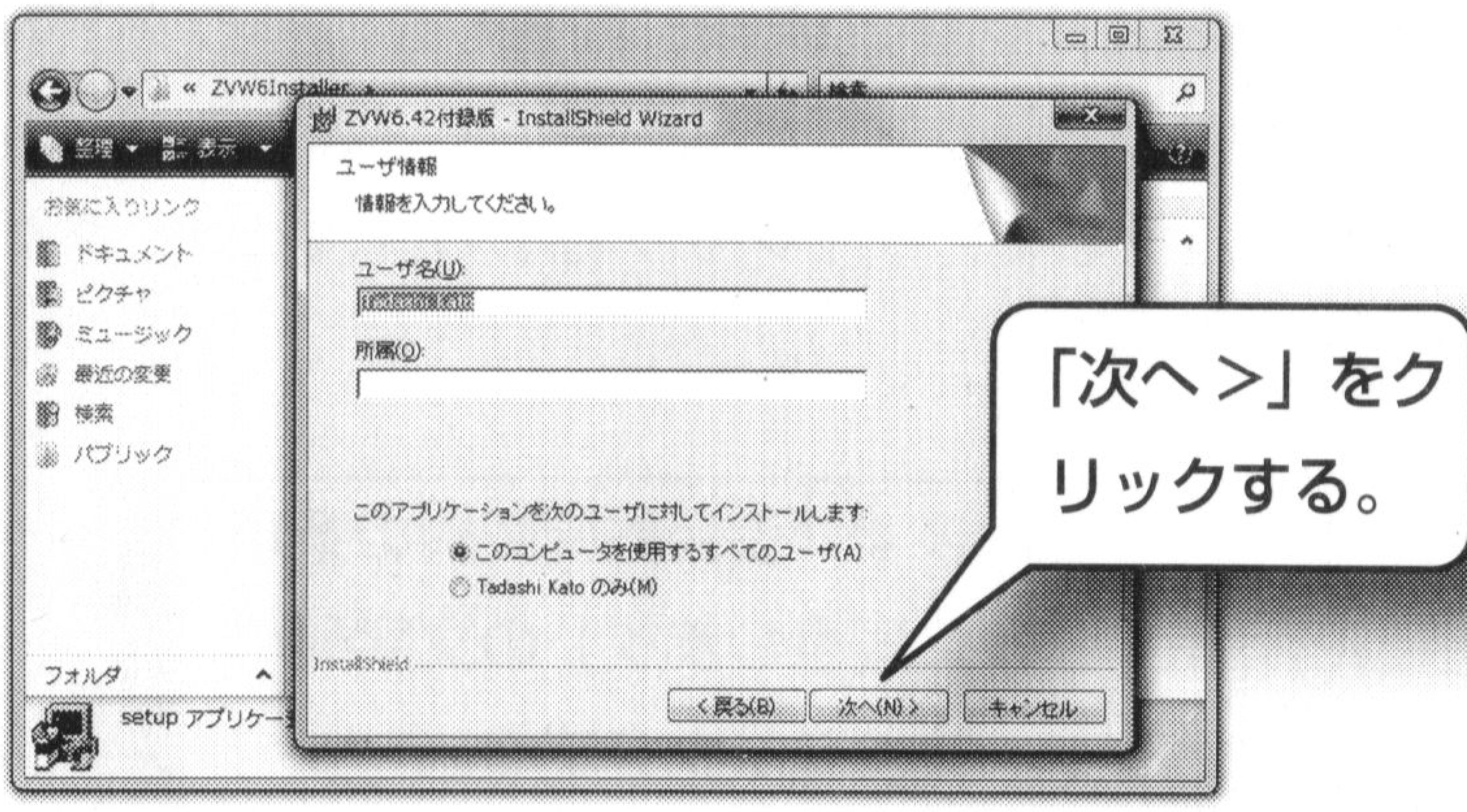


図2-5　ユーザ情報を入力する

⑥セットアップタイプを選ぶ画面ですが、ここではデフォルトの「標準」を選び、「次へ>」をクリックします。

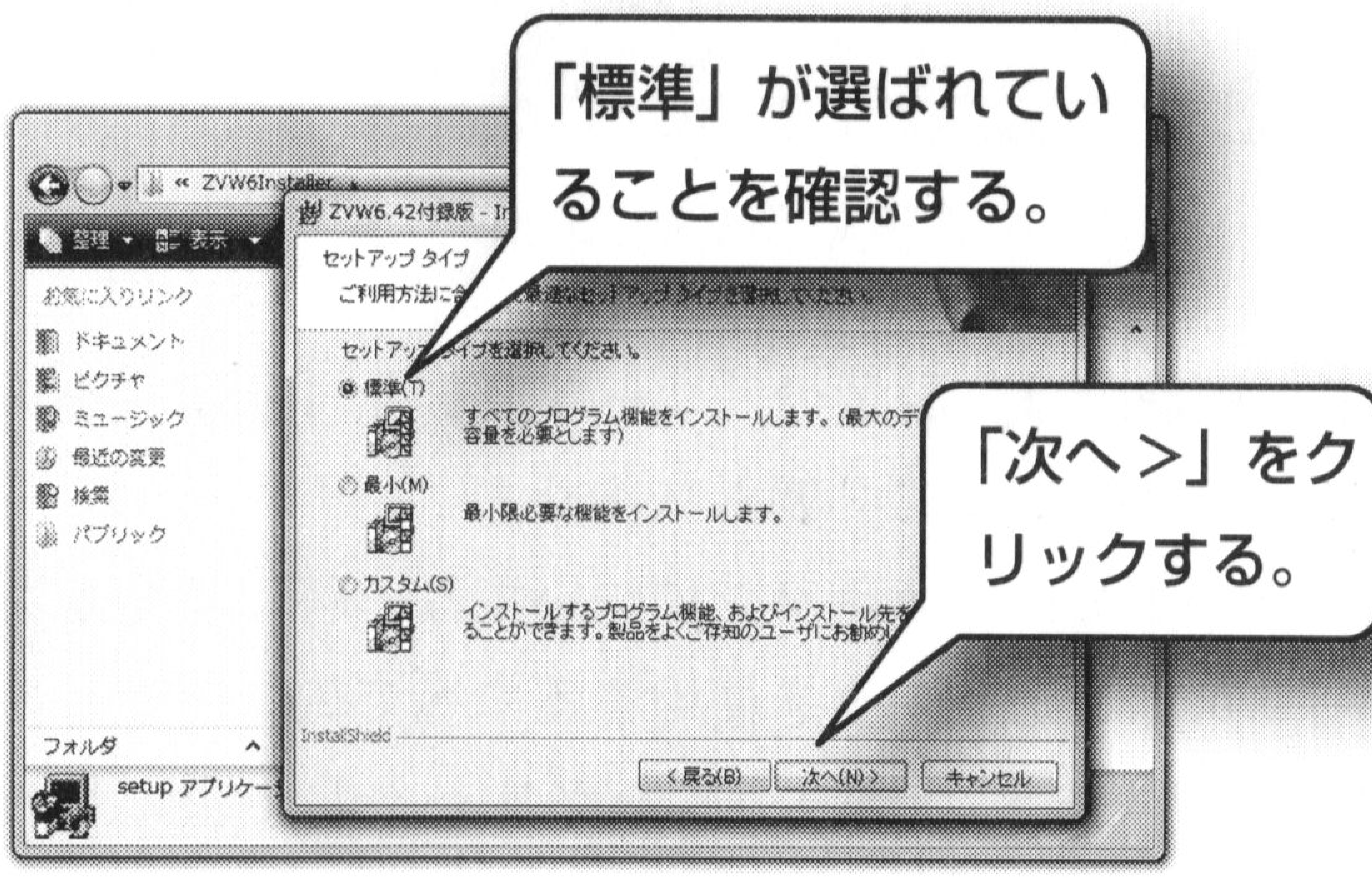


図2-6　セットアップタイプを選ぶ

⑦図2-7で各種の設定内容を確認したら、「インストール」をクリックします。

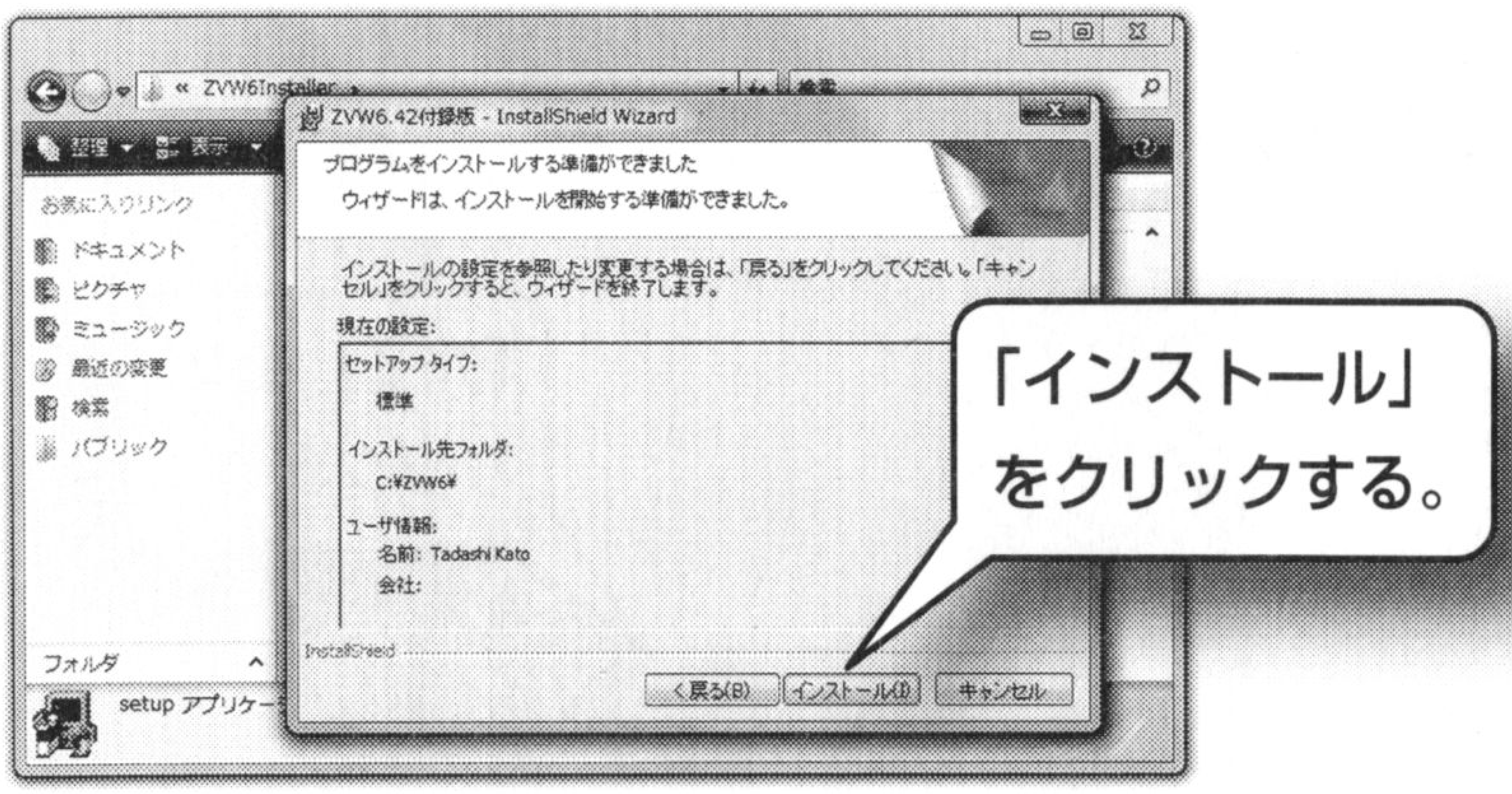


図2-7　インストール内容を確認する

⑧パソコンの環境によっては、図2-8の状態で10秒以上待たされることがありますが、トラブルではありません。やがて、インストール作業が自動的に始まります。

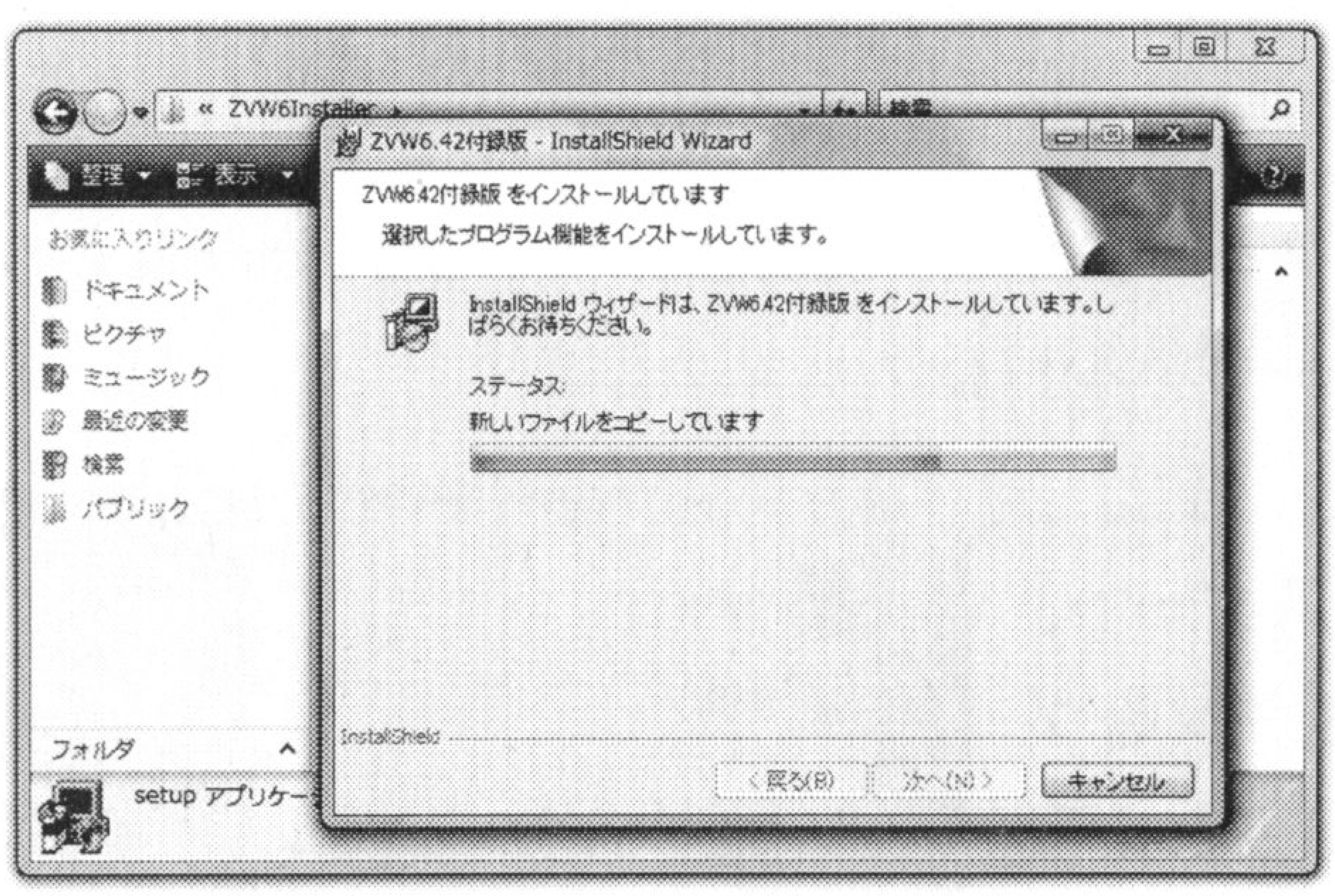


図2-8　インストール開始

⑨インストールが完了しました。「完了」ボタンをクリックしてください。

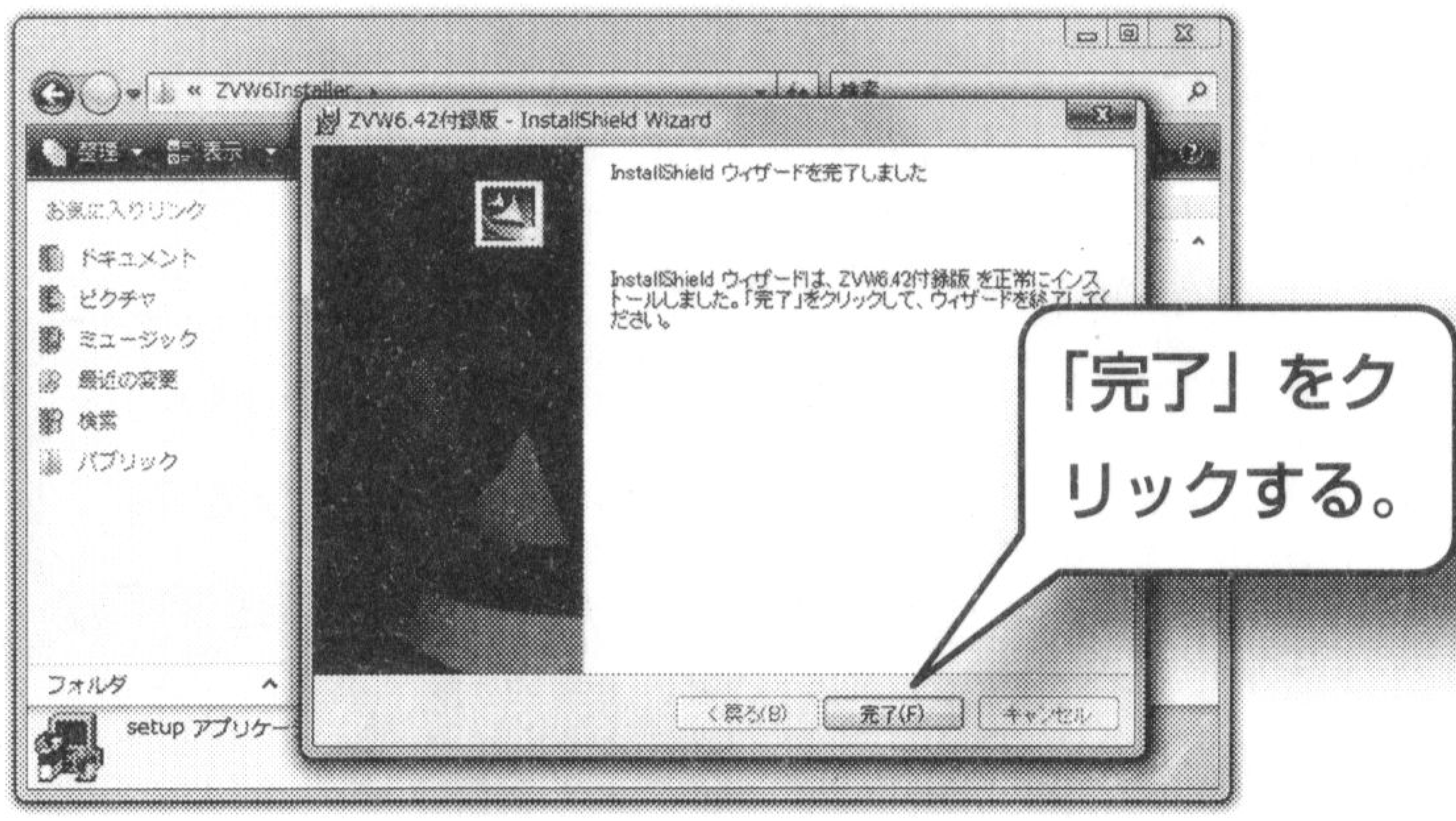


図2-9　インストール完了

2.2　アンインストールの手順

インストールとは逆に、Z-Visionをアンインストールする手順についても触れておきます。

①「スタート」ボタンをクリック、次にその右上の「コントロールパネル」をクリックします。



図2-10　コントロールパネルを開く

②コントロールパネルの中で、プログラムアイコンを探し、「プログラムのアンインストール」という文字列をクリックします。



図2-11　「プログラムのアンインストール」を開く

③プログラムの一覧から「ZVW6.42付録版」を探し、これをクリックします。次に、プログラムと機能ウィンドウの上方にある「アンインストール」という文字列をクリックします。

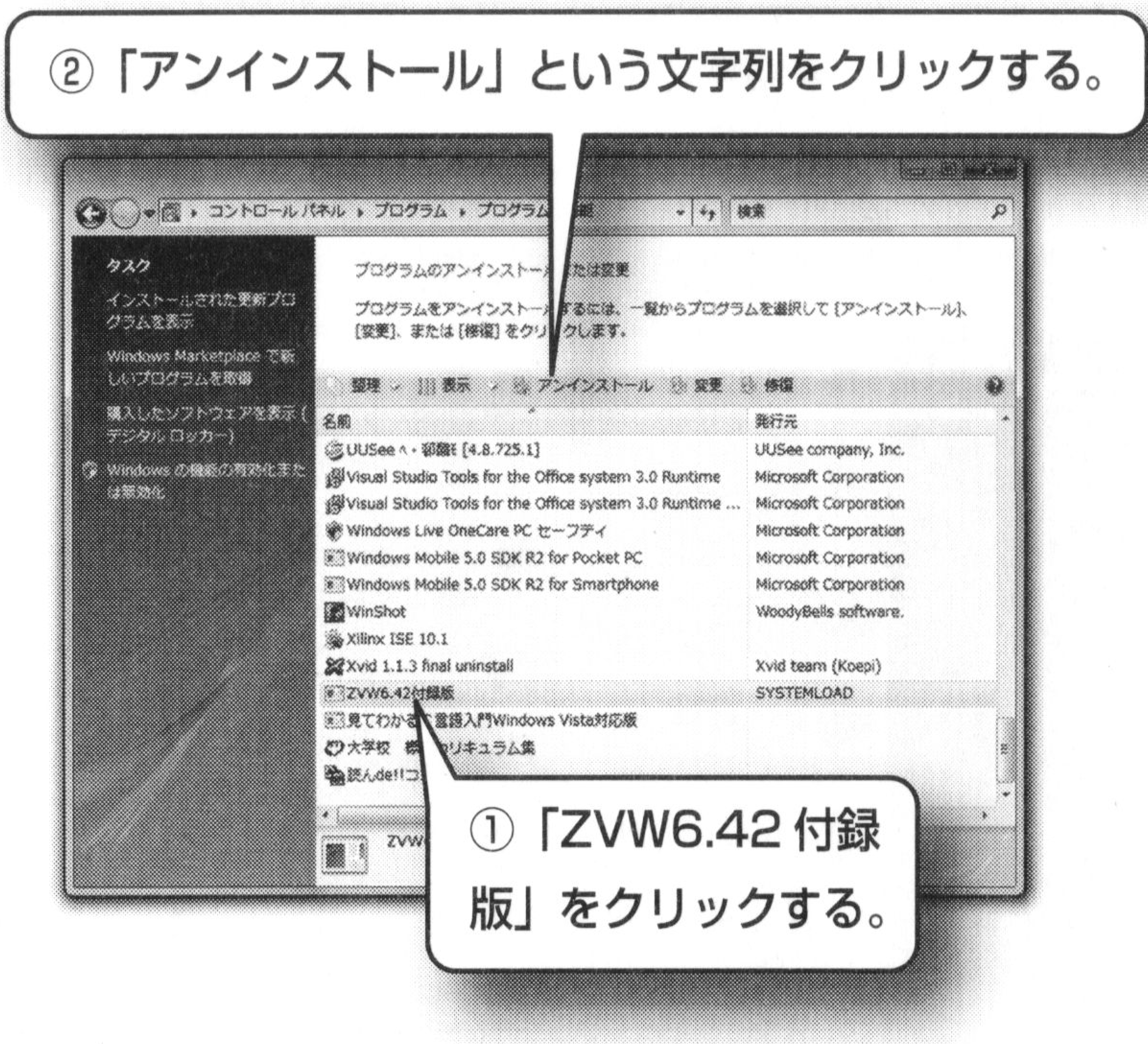


図2-12　「ZVW6.42付録版」を選ぶ

Windows Vistaでは、ここで「ユーザアカウント制御」という警告ウィンドウが表示されますが、「→許可」のボタンをクリックして、先に進んでください。

図2-13の状態でしばらくすると、アンインストールが完了しますので、プログラムと機能ウィンドウを閉じます。また、ディスクドライブからCD-ROMを取り出しておいてください。



図2-13　アンインストールが始まる

2.3　Z-Visionを起動する

Z-Visionは、米マイクロソフト社がWindowsを販売する以前から、DOS/Vパソコンで使われていました。

MS-DOS（Microsoft Disk Operating System）の上で走っていたZ-VisionはWindows用のアプリケーションソフトとして作り直されましたが、ウィンドウの形をしているという点以外、大幅な変更はありません。

そのため、デザインそのものは、いたってシンプルです。

①「スタート」ボタン→「すべてのプログラム」とクリックすると、その上方に「ZVW6」というアイコンが現れますので、これをクリックします。

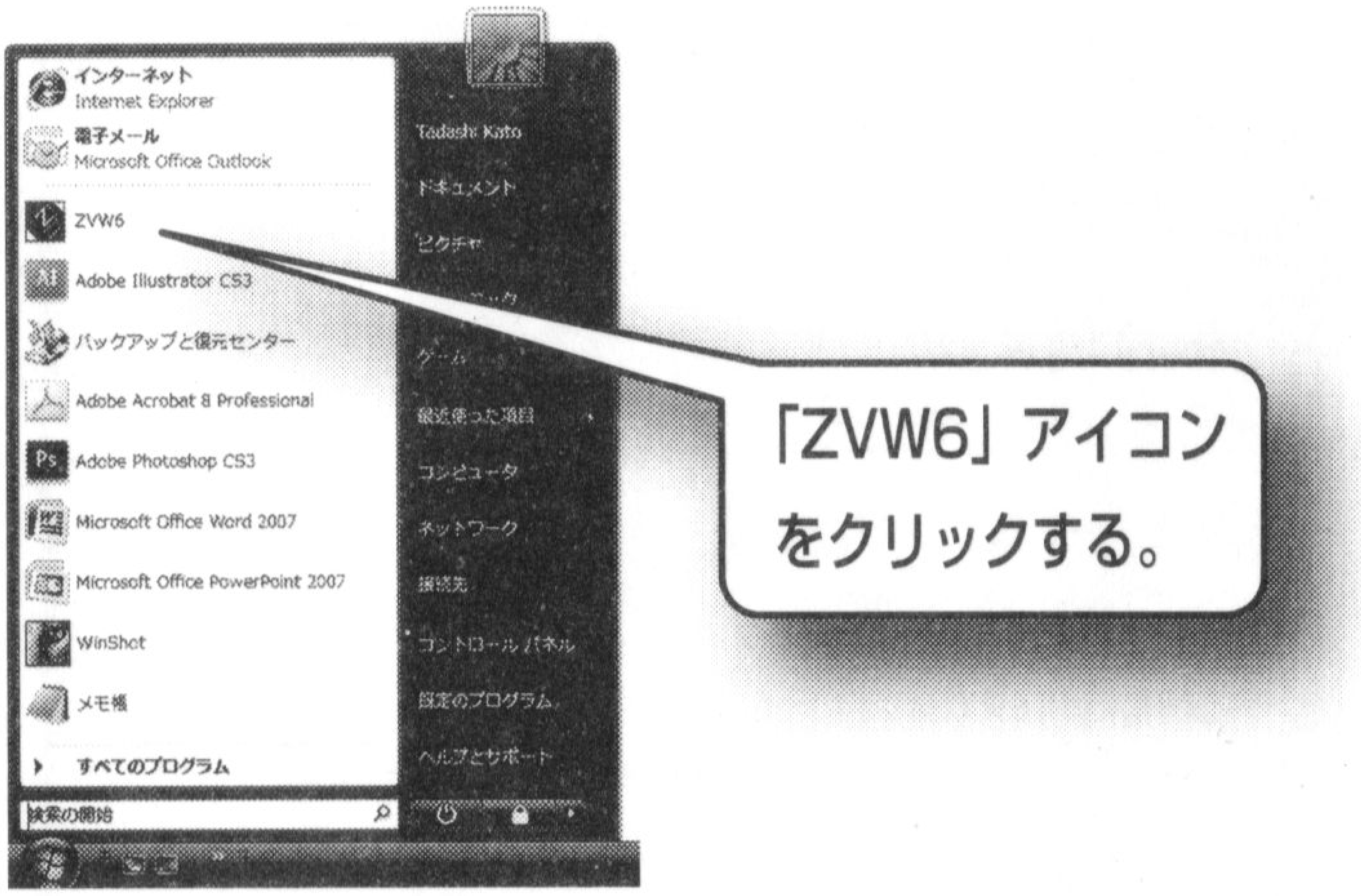


図2-14　Z-Visionを起動する

②初期画面が現れます。この状態で第3章へ進んでください。

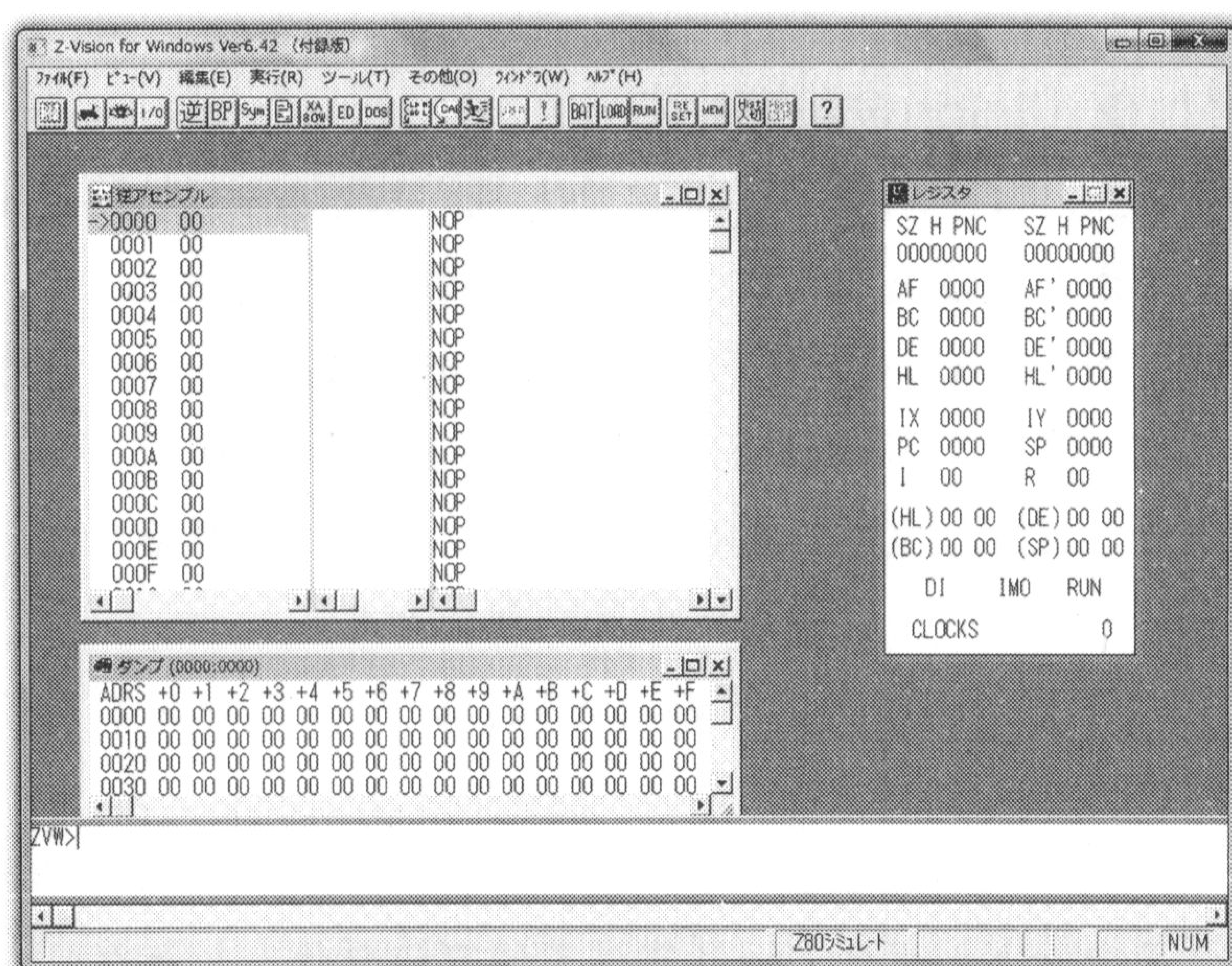


図2-15　Z-Visionの初期画面

第3章　プログラムを入れてみる

Z-Visionが起動しました。これから、Z80の内部に入っていきますが、まだプログラムがありません。そのため、パソコンには、いくつかのウィンドウが現れてはいるものの、その中の数字はゼロ（0）ばかりです。

Z-Visionの機能を紹介していく上で、プログラムは不可欠です。なぜなら、シミュレートすべきプログラムがZ-Visionに導入されている場合と、そうでない場合とでは、表示内容が著しく異なるからです。

Windowsアプリケーションでおなじみの、メニューバーやボタンアイコンが並んでいますが、ゼロばかりでは楽しくありません。

そこで、本章では簡単なテストプログラムを入力し、アセンブリ言語の構文に慣れていただくことにします。

3.1　テストプログラムを作る

これから、アセンブリ言語を用いて、テストプログラムを入力していきます。

繰り返しになりますが、CPUが実行する命令や演算する値はすべて数字の列で、これを**機械語**と言います。しかし、CPUが理解するように、人間が数字だけでプログラムを作成するのはきわめて困難です。

そこで、数字の意味を、英単語やそれらを組み合わせた略語で表現することによって、ユーザがプログラムできるようにしたのが**アセンブリ言語**です。アセンブリ言語の構文については、3.2節で述べます。

①先ほど起動したZ-Visionで、「ED」ボタンをクリックします。EDはエディタ（editor）の略語で、厳密には、コンピュータ上で文字情報を入力・編集するためのテキストエディタ（text editor）のことを指します。

Windowsでは、標準で備わっているメモ帳が起動します。

②初めてZ-Visionでメモ帳を起動すると、図3-1のように、「ファイル.asmが見つかりません。新しく作成しますか？」と聞いてきますので、「はい」をクリックします。

ここに表示された.asmは、アセンブリ言語で記述されたプログラムを保存するときの拡張子です。



図3-1　テキストエディタを起動する

③ファイル名が「無題」のウィンドウが現れ、カーソル（|）が点滅しています。

これから、シミュレータを動作させるための、テストプログラムを入力していきます。CPUの中にいくつか入っているレジスタと呼ばれる小さなメモリの値を、次々に変えていくだけのシンプルなプログラムです。

入力するにあたっては、エラーを発生させないための注意すべき点が、いくつかありますので、プログラムリストと個々に注意事項を以下に述べます。図3-2も合わせてご覧ください。



図3-2　テストプログラムを入力する

まず、慣れるまでは、使用する文字をすべて半角英数とします。フォントは任意です。なるべく数字の「0」とアルファベットの「O」が識別しやすい方がよいでしょう。また、くれぐれも全角のスペースを使わないように。

文字列の右端まで入力したら、Enterキーを押します。また、以下のリストは解説文を追加したものですので、㊟を付けた文字列は実際には入力しません。

```
;                           ㊟左端の「；」はセミコロン
;TEST PROGRAM
;
        LD      A,1         ㊟左端でTabまたは半角スペース
        LD      B,1
        LD      C,1
        LD      D,1
        LD      E,1
        LD      H,1
        LD      L,1
;
LOOP:   INC     A           ㊟「LOOP」はアルファベットのオーが2つ
        INC     B
        INC     C
        INC     D
        INC     E
        INC     H
```

メモ帳のデフォルトのフォントは、MS ゴシック。

```
            INC         L
;
            EX          AF,AF'
            EXX
            JP          LOOP
;
            END                     ㊟これが最終行
```

④最終行まで入力したら、ファイルメニューから「名前を付けて保存」を選びます。

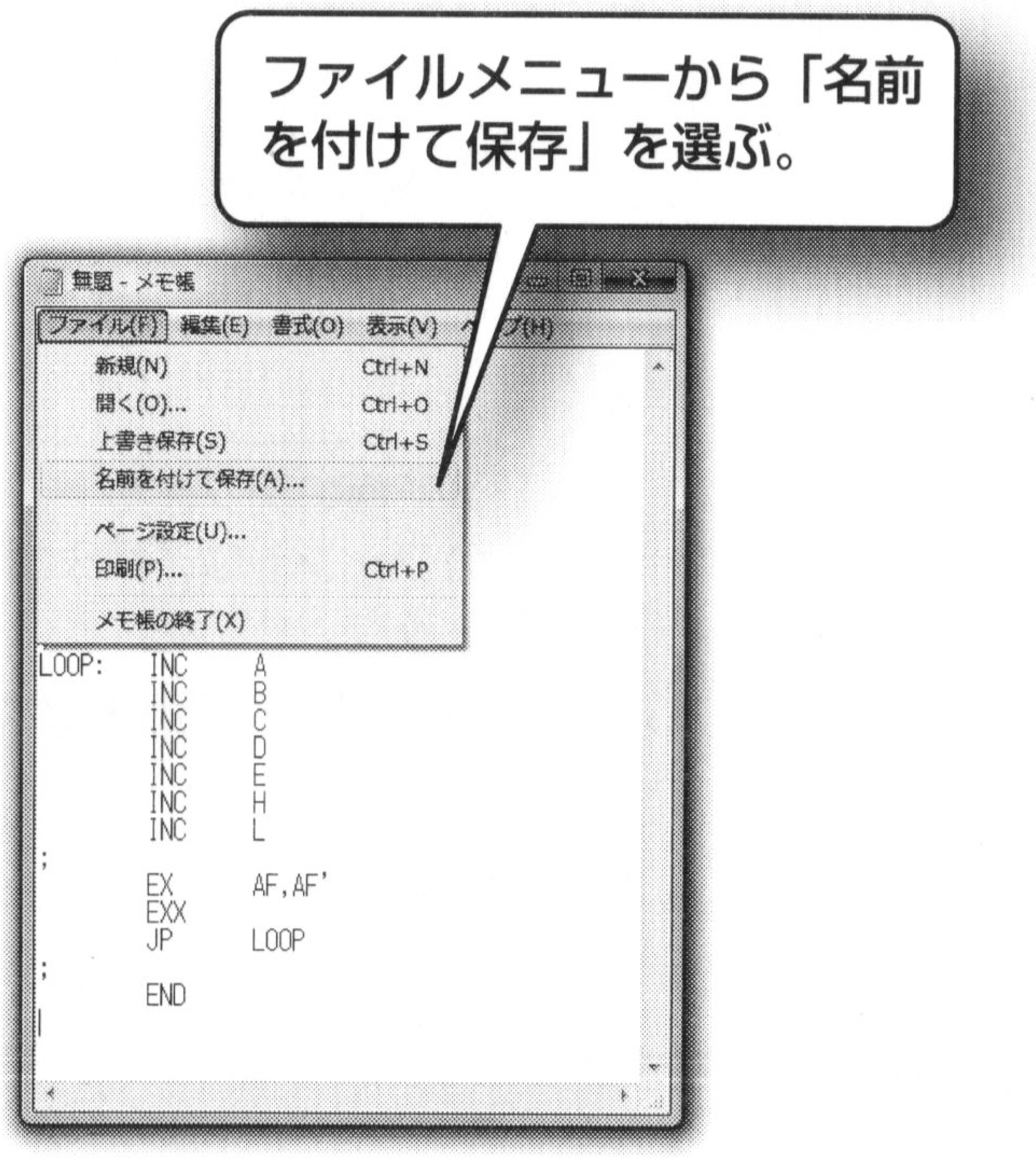


図3-3　プログラムリストを保存する

⑤「名前を付けて保存」ダイアログボックスが開きますので、まずCドライブ上に保存用フォルダをつくります。「新しいフォルダ」という文字列をクリックすると、フォルダアイコンが現れます。



図3-4　Cドライブに保存用フォルダをつくる

Z-Visionは、指定したフォルダの中で、各種の変換作業を行いますので、メモリ容量に多少の余裕が必要となります。

数メガバイトもあれば十分ですが、読者の皆さんが今後多くのプログラムを作成されることを考慮しますと、パソコンシステムの中で常に接続されているCドライブであれば確実です。特にフロッピーディスクドライブは、トラブルの元になります。

⑥ここでは筆者の名前を利用して「Z80_KATO」と入力しました。

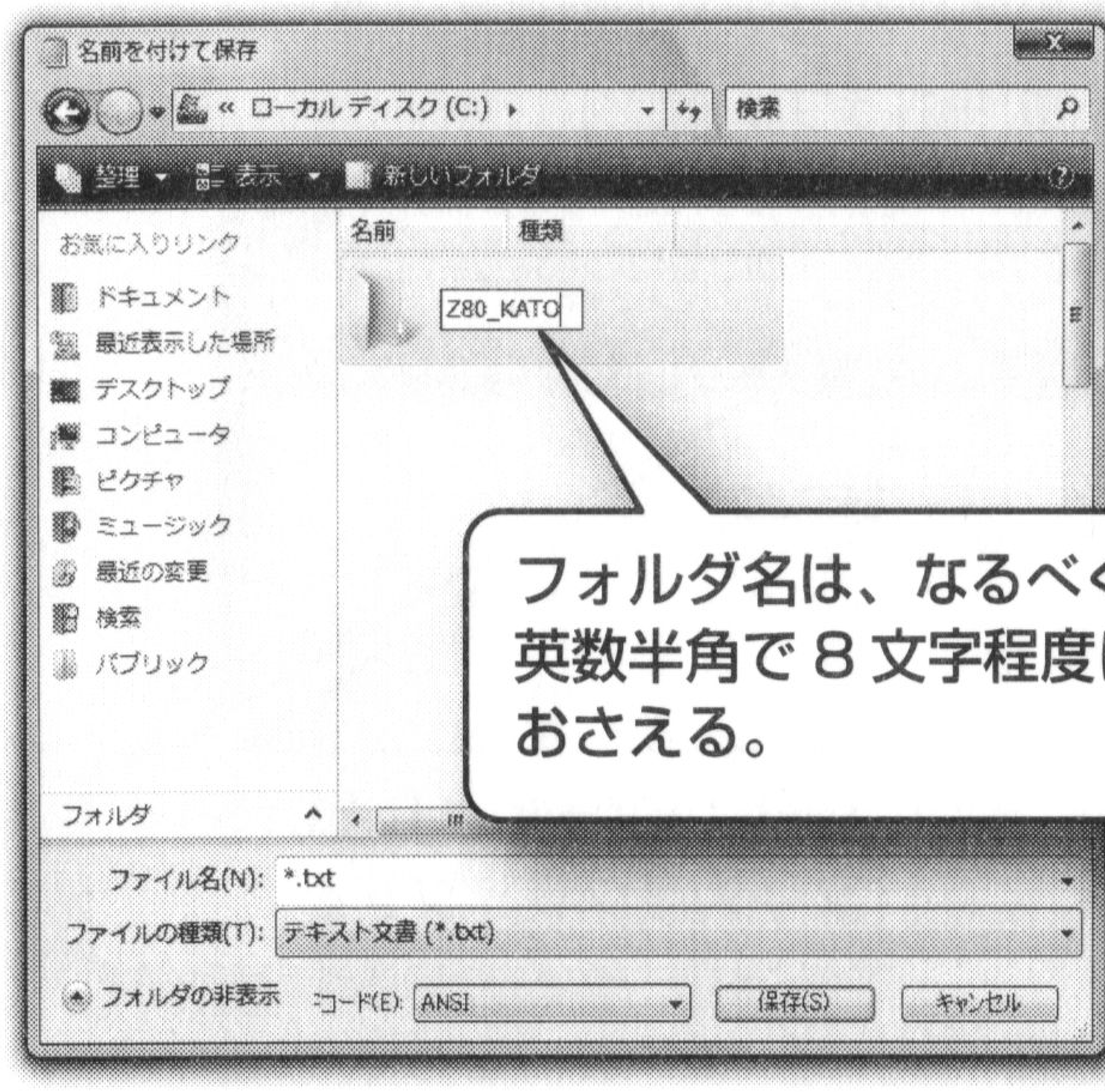


図3-5　フォルダ名を入力する

ところで、フォルダ名やファイル名を決めるとき、本来ならばWindowsの中ですから、文字数をあまり気にしなくても任意に選ぶことができますが、Z-Visionを使うときは、なるべく英数半角で８文字程度までの文字列を使ってください。

Z-Visionは、かつてMS-DOS上で使われていました。基本設計は、現在も大きく変わってはいません。そのため、ダイアログボックスによっては、ファイル名などの表示文字の一部が省略される場合があるのです。

筆者が教育現場でひんぱんに経験してきたことですが、似たようなフォルダ名やファイル名が複数個あると、しば

しば判別に戸惑い、ときには最新データを誤って失うこともあります。

⑦さて、保存用フォルダを開き、ここにプログラムリストを保存するのですが、デフォルトの拡張子.txtではZ-Visionで走らせることができません。

　ダイアログボックスの下の「ファイルの種類」が「テキスト文書」になっていますので、この文字列をクリックします。すると、「すべてのファイル」という文字列が見えますので、さらにこれをクリックします。

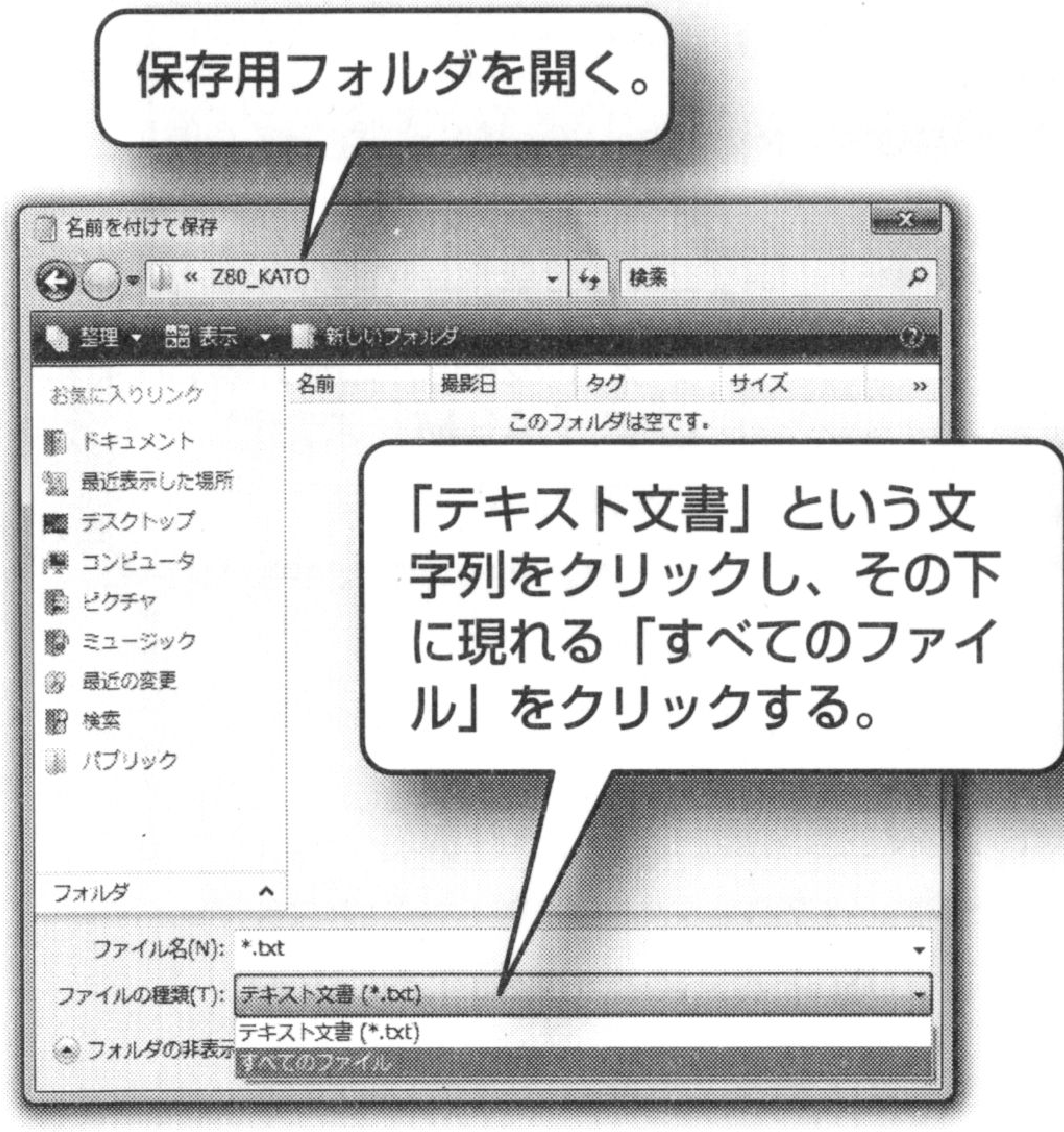


図3-6 「ファイルの種類」を変更する

⑧「ファイル名」のテキストボックスに、読者の希望するファイル名を入力します。ここでは、「TEST.ASM」としました。

拡張子の.ASM（または.asm）だけは変更しないようにしてください。別の拡張子を使うと、機械語への変換ができなくなります。なお、拡張子には大文字と小文字の区別はありません。

入力が完了したら、「保存」ボタンをクリックし、ダイアログボックスを閉じます。また、メモ帳も必要ありませんので、閉じてください。



図3-7　ファイル名を入力する

3.2　アセンブリ言語の構文

アセンブリ言語でプログラムを記述するときには、いくつかの決まりごとにしたがう必要があります。図3-8は、テストプログラムの主要な部分を抜き出して、コメントを追加したものです。

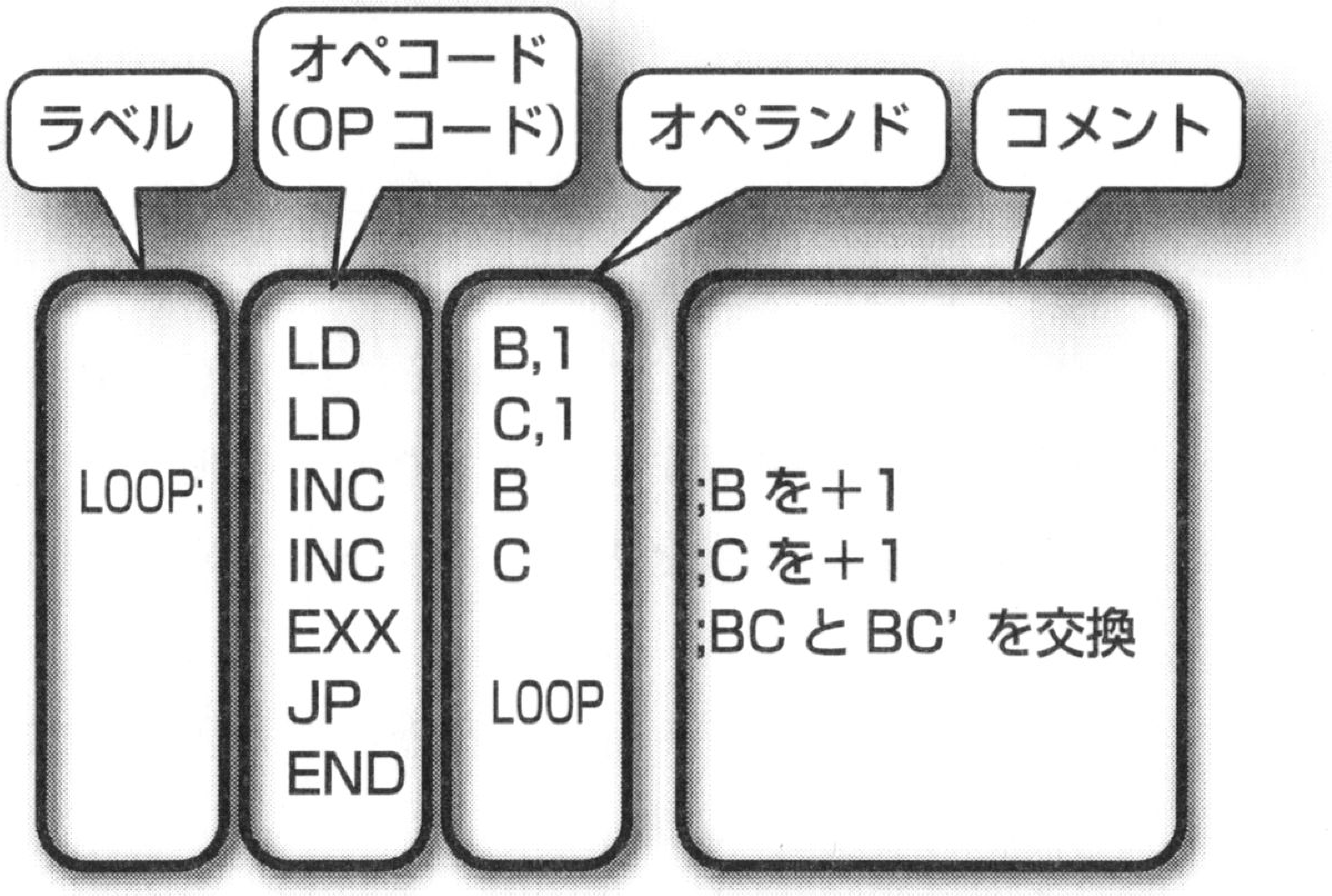


図3-8　アセンブリ言語の構文

①各行の構成

アセンブリ言語による1行のプログラムは、ラベル、オペコード（ニーモニックとも呼ぶ）、オペランド、コメントの4つで構成されます。

また、1行には命令が最大1個しか記述できず、複数行にわたって記述することができません。ただし、空白行やコメントについては制限はありません。

②ラベル欄（label）

アドレス値を指定するには、たとえば16進数で1234Hと記述しますが、つくった本人はともかくとして、他者がこの数字を見ても、何の意味なのかがわかりません。

そこで、言葉としての意味を持つラベルを使うことによって、直感的にわかりやすいプログラムをつくることができます。

また、ジャンプ先のアドレス値は、アセンブリ言語で記述しているときには、まだ決まっていないことが多く、予測することは困難です。そこで、3.4節に述べる「アセンブラ」という機械語に翻訳するソフトウェアを使うことによって、自動的アドレス値を決めてくれる仕組みを採ったのです。

ラベルは、第1カラム（1行の最左端）から書きます。そして通常は、ラベルの最後に「:」(コロン）を付けます。ラベルとして使える文字は、AからZのアルファベット、0から9の数字と、次の特殊記号です。

! # $ % & ? @ 「 」 ^ _ { }

なお、ラベルの先頭に数字を置くことはできません。また、以下の記号や文字列は予約語と呼ばれ、ラベルとしては使用できません。

■CPUのレジスタ名

A B C D E H L R I

AF BC DE HL IX IY PC SP

■演算子

$　AND　EQ　GE　GT　HIGH　LE　LT

LOW　MOD　NE　NOT　OR　SHL　SHR　XOR

■ニーモニック（第7章で主なものを紹介）

Z80のすべてのニーモニック、つまりアルファベットを用いた命令語のことです。

■疑似命令（3.3節参照）

；　EQU　ORG　END　など

なお、ラベルは30文字まで可能ということになっていますが、プログラムリストの見やすさやバグを防ぐ上でも、8文字以内をお薦めします。

③オペコード欄（operational code）

ニーモニック（mnemonic）とも呼ばれ、2文字から4文字で表記されます。CPUが実行する機械語に対応する命令語です。

④オペランド欄（opeland）

オペランドとは、命令語によって操作されるレジスタやアドレス値のことです。

Z80では、オペランドがない命令と、パラメータ1個、パラメータ2個の3通りがあります。たとえば、「LD C,1」のようにパラメータが2個のときは「,」（カンマ）で区切ります。

⑤コメント欄（comment）

１行の中に、疑似命令の「;」（セミコロン）があると、それよりも右側の文字列は、すべてがコメント（説明文などの文字列）として扱われます。つまり、行の左端にセミコロンがあれば、行全体がコメントとなります。

⑥定数の使い方

数値定数、文字列定数、特殊定数の３種類の定数が使用できますが、ここでは本書でひんぱんに使う数値定数について述べます。

数字から始まる一連の文字列を数値定数と言い、末尾に**基数表現文字**というアルファベットを付けることによって、「進数」を区別しています。

また、数値定数は２バイトで表される数値に制限されています。つまり、10進数で０から65535までの範囲の数であることが必要です。

■10進数（デシマル）

数字の０～９で構成された文字列です。末尾にDを付加して10進数であることを明示することがありますが、通常は省略します。

■16進数（ヘキサデシマル）

０～９、Ａ～Ｆで構成された文字列で、末尾にHを付けます。またAFHのように、アルファベットから始まる数値の場合は、先頭に0（ゼロ）を付けて0AFHとします。

■2進数（バイナリ）

0または1で構成された文字列で、末尾にBを付けます。

3.3 アセンブリ言語の疑似命令

アセンブリ言語のプログラムでは、ニーモニック以外に、直接機械語には翻訳されない、アセンブラに対する指示文が含まれます。これを疑似命令と言います。

CPUにしてみれば、本来は不要なものですが、人間にとっては、疑似命令なしに効率的なプログラム開発はできません。本書で用いる主な疑似命令を以下に紹介します。

①；

前節で少し触れましたが、コメント記号と呼ばれ、コメントの開始を表します。1行の中で、セミコロン以降は、コメントとみなされます。

ニーモニックのない行の場合は、左端に「;」を入れます。また、ニーモニックのあとならば、スペースキーかTabキーで空白を入れてから、「;」を入れます。「;」以降は、全角の漢字や仮名など、表現は自由です。

アセンブリ言語の欠点は、その単純過ぎる表現です。そのため数日も経つと、作った本人ですら内容を思い出せないくらいです。コメントを入れるのは確かに面倒ですが、時間が経つにつれてその価値に気づくでしょう。

②END

エンドと言います。アセンブラに、「プログラムリストが

以上で終わった」ということを伝える疑似命令です。ASMファイルのすべての記述が終わったあとに、必ず付けてください。この行がないと、アセンブラを起動したときにエラーが発生します。
「END」の前の空白は、特に必要ありませんが、見やすさの点で、Tabキーを入れておくことをお薦めします。

　また、この命令文のあとは、何が書いてあってもアセンブラは無視しますので、自由にコメントや解説文を書くことができます。

③EQU

　イコールと言います。たとえば、

```
TEISU   EQU     100
```

と記述して、ラベルに数値を割り当てます。

　この場合は、プログラムの中に

```
        LD      A,TEISU
```

があると、アセンブラは

```
        LD      A,100
```

と解釈してアセンブル（機械語に翻訳すること）作業を続けます。

　プログラムの中で、複数の定数をイコール文で定義しておけば、記憶しやすい文字列を選ぶことができるので、効率よくプログラム開発ができます。

　たとえば、I/Oポートのアドレス表現によく使います。

　なお、テストプログラムで「LOOP:」と入力したとき

のアドレスの値とは、使い方が異なります。コロン「:」は、メモリのアドレスを意味する記号で、これがないラベルはイコール文だけです。

④ORG

オリジンと言い、アセンブルの開始番地を指定します。もし、この命令がないときは、アセンブラは0000H番地から順に、機械語の数列を作成していきます。

テストプログラムでは、このオリジン命令を省略しているため、0000Hからロードされたのです。

たとえば、

```
        LD      A,1
```

だけのプログラムをアセンブルすると、リストファイルは

```
0000    3E 01           LD      A,1
```

となりますが、この命令文の前に、「ORG　8000H」を付けると、

```
                        ORG     8000H
8000    3E 01           LD      A,1
```

となります。リストファイルについては、次の3.4節で詳しく述べますが、アドレスや機械語、アセンブリ言語が1行にレイアウトされたファイルのことです。

⑤DB（参考）

デファインバイトと言います。

メモリの内容を1バイトごとに、あらかじめ定義しておくときに使用します。たとえば、

```
        ORG     8000H
DATA:   DB      3,2,1
```

と書けば、メモリの中は次のように設定されます。

アドレス	データ
8000H	03H
8001H	02H
8002H	01H

つまり、先頭にラベルを付けて、あとは順番に配置されるのです。

⑥DW（参考）

デファインワードと言います。

「DB」と似ていて、メモリの内容を、2バイトごとに定義することができます。

```
        ORG     8000H
DATA:   DW      3,1234H,10H
```

と書けば、メモリの中は次のように設定されます。

アドレス	データ
8000H	03H
8001H	00H
8002H	34H
8003H	12H
8004H	10H
8005H	00H

3.4　アセンブラを起動する

アセンブリ言語でプログラムリストを作成したら、これを機械語に変換しなくてはなりません。

①Z-Visionのウィンドウで「XA80W」ボタンをクリックすると、XA80Wウィンドウが現れます。

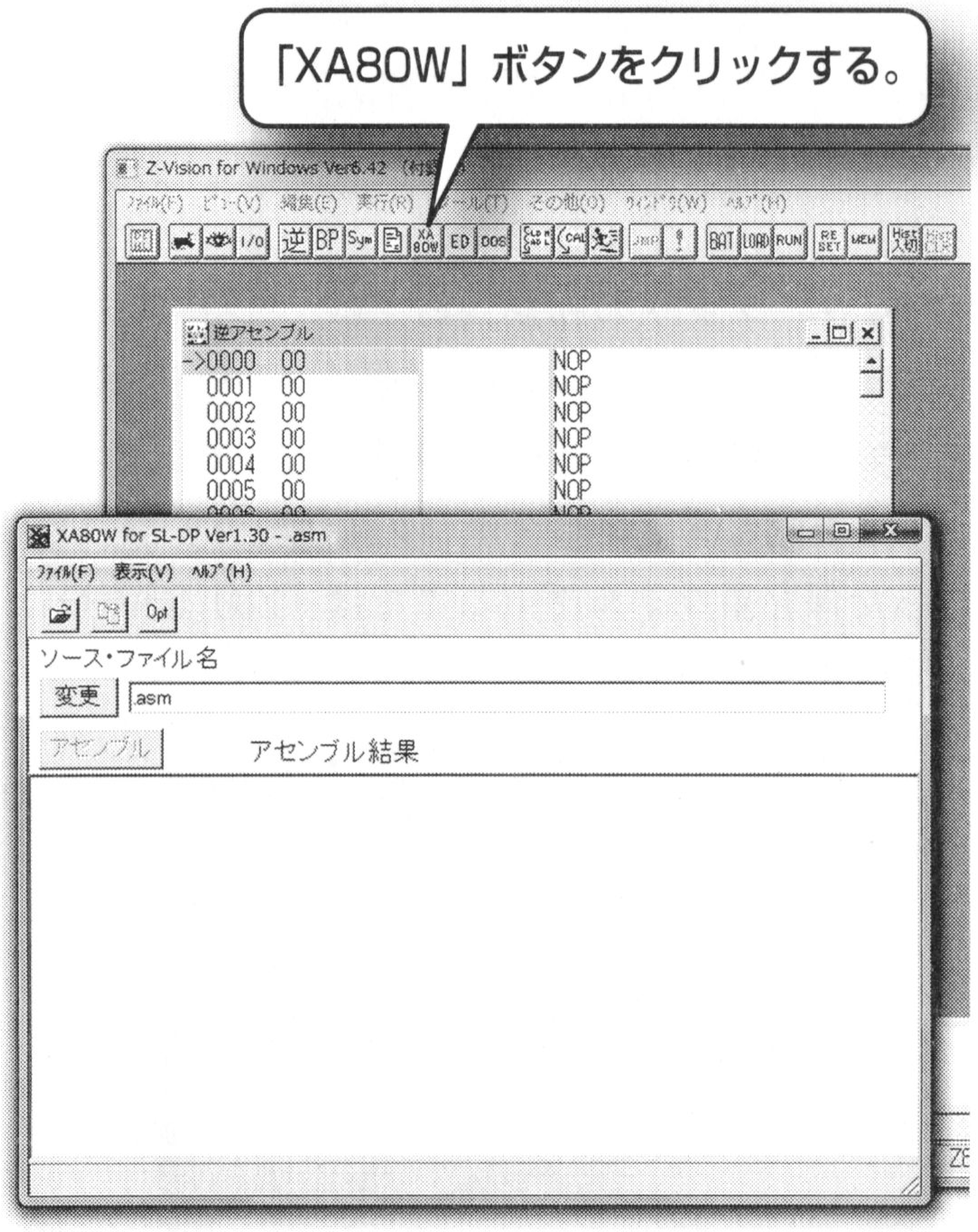


図3-9　クロスアセンブラを起動する

ところで、XA80Wウィンドウは、ダイアログボックスではありません。Z-Visionとは別の独立したソフトウェアです。「XA80W」は商品名ですが、これには「Windows版のZ80用クロスアセンブラ」という意味があります。

アセンブラは、アセンブリ言語を機械語に翻訳するソフトウェアのことです。また、クロスアセンブラはアセンブラの一種で、実際に動かすCPU（ここではZ80）とは別のCPU（たとえばPentium）の上で翻訳作業をするソフトウェアのことを指します。

②XA80Wウィンドウの初期画面では、「ソース・ファイル名」のテキストボックスが、拡張子.asmだけになっていますので、「変更」というボタンをクリックします。

ファイルを開くダイアログボックスが現れますので、3.1節で作った保存用フォルダ（ここでは「Z80_KATO」）を探し、開きます。



図3-10　保存用フォルダを開く

よく見かけるミスとしては、メモ帳で保存する際に、あわてて標準の拡張子である.txt（または.TXT）で保存した場合です。そのときは、このダイアログボックスに、ファイルのアイコンが現れませんので、ご注意ください。

③先ほど保存した「TEST.ASM」を選び、「開く」をクリックすると、ダイアログボックスが閉じて、XA80Wウィンドウに戻ります。

XA80Wウィンドウの「ソース・ファイル名」の欄に、開いたファイル名が現れましたら、「アセンブル」ボタンをクリックします。

このときのファイル名には、Cドライブを意味する「C:」や、フォルダを意味する「¥」が混じっていることがあります。少し難しく見えますが、そのファイルがどこに保存されているかを正確に表しているので、慣れるようにしましょう。

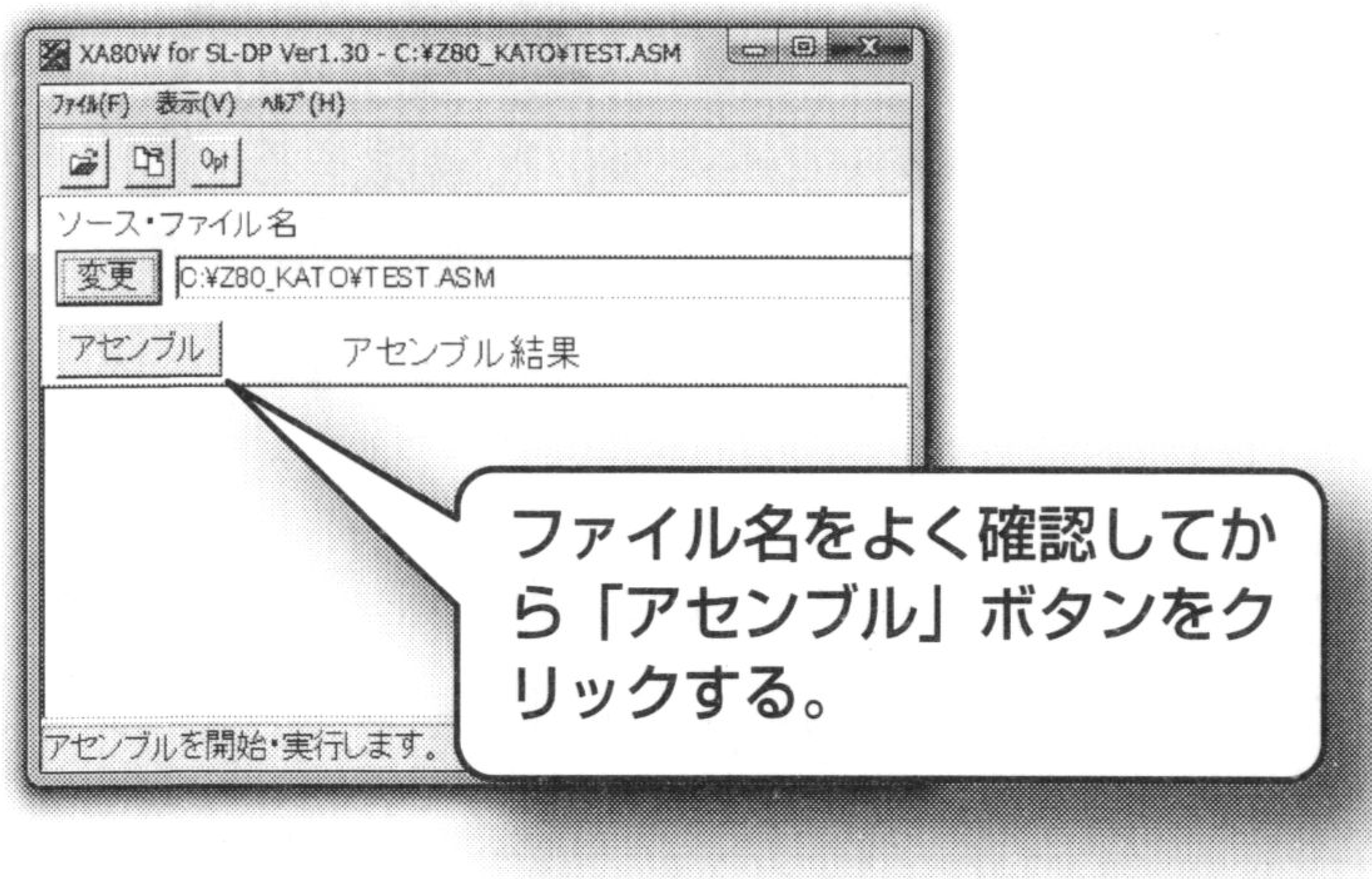


図3-11　ソース・ファイルをアセンブルする

④アセンブル情報というウィンドウが現れます。「警告&エラー情報」という欄で、「Warnings」と「Errors」の右側に「0（ゼロ）」が記載されていれば、プログラムリストにエラーはありません。「OK」ボタンをクリックすると、ウィンドウが閉じます。特にWarningsは、存在しないニーモニックを記述したときに、よく発生します。

たとえ、エラーがわずか1つであっても、このあとの実験で正常に動作しない場合があります。その場合は、「OK」ボタンをクリックしたのち、めんどうでも、メモ帳でプログラムリストの入力ミスを修正後、保存し直してください。

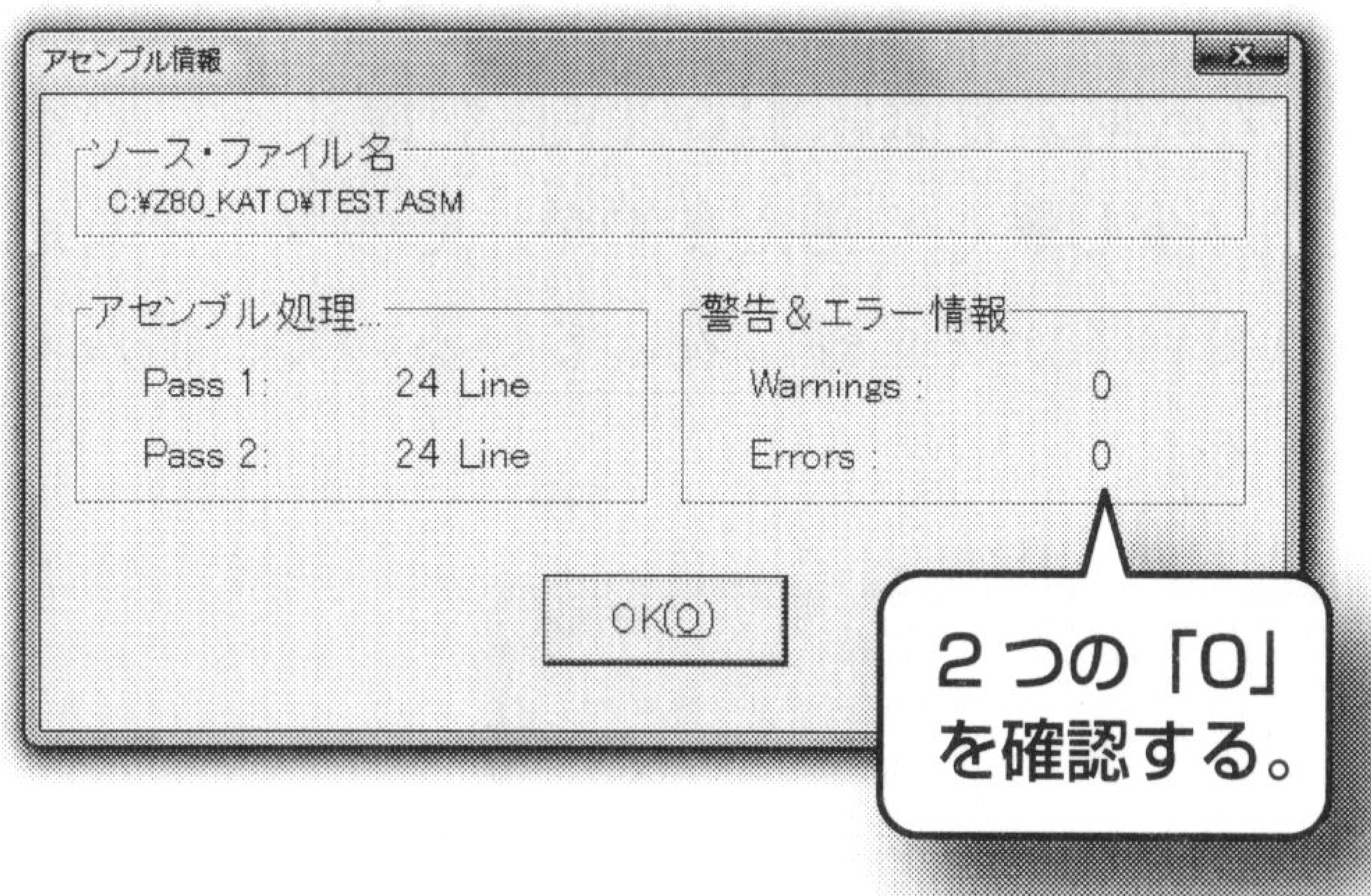


図3-12　アセンブルの結果を確認する

⑤正常に終了していれば、「XA80W」ウィンドウの下方に、アセンブル（翻訳作業）の経過メッセージや、「Warnings」と「Errors」がなかったという情報が表示されます。

ファイルメニューから「アプリケーションの終了」か、ウィンドウ右上の×印（閉じるボタン）をクリックして、このウィンドウを閉じます。

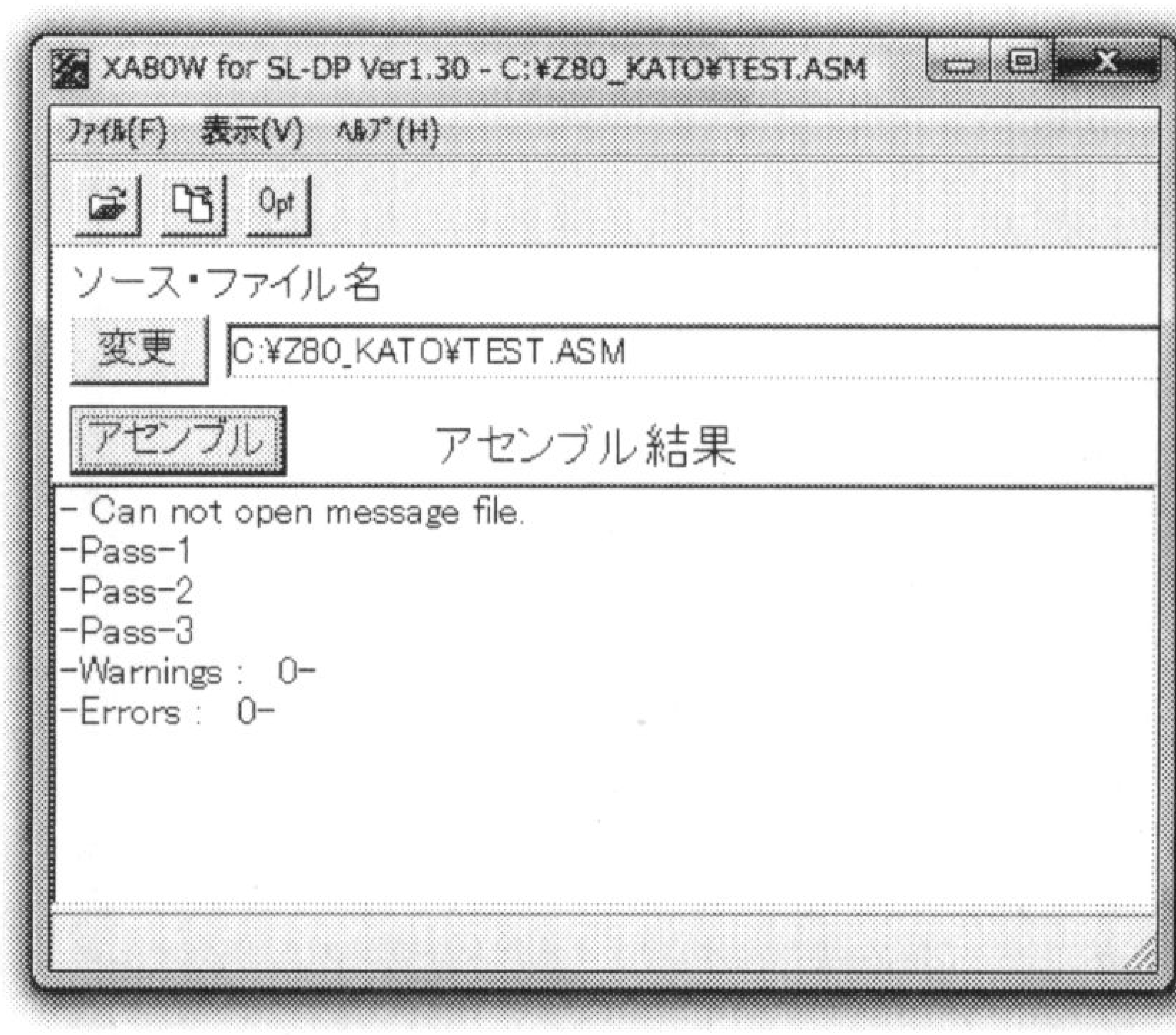


図3-13　アセンブル完了

以上の操作によって、「TEST.ASM」というソース・ファイル（ASMファイル）の他に、「TEST.HEX」という機械語のファイルが生成されました。拡張子が.HEXなので、一般にヘキサファイルと呼びます。

⑥実は、XA80Wの仕事は、ヘキサファイルをつくることだけではありません。保存用フォルダをのぞいてみてください。ソース・ファイルを除いて5つのファイルが追加されていることがわかります。

いずれのファイルもメモ帳で開くことができます。実際にやってみましょう。

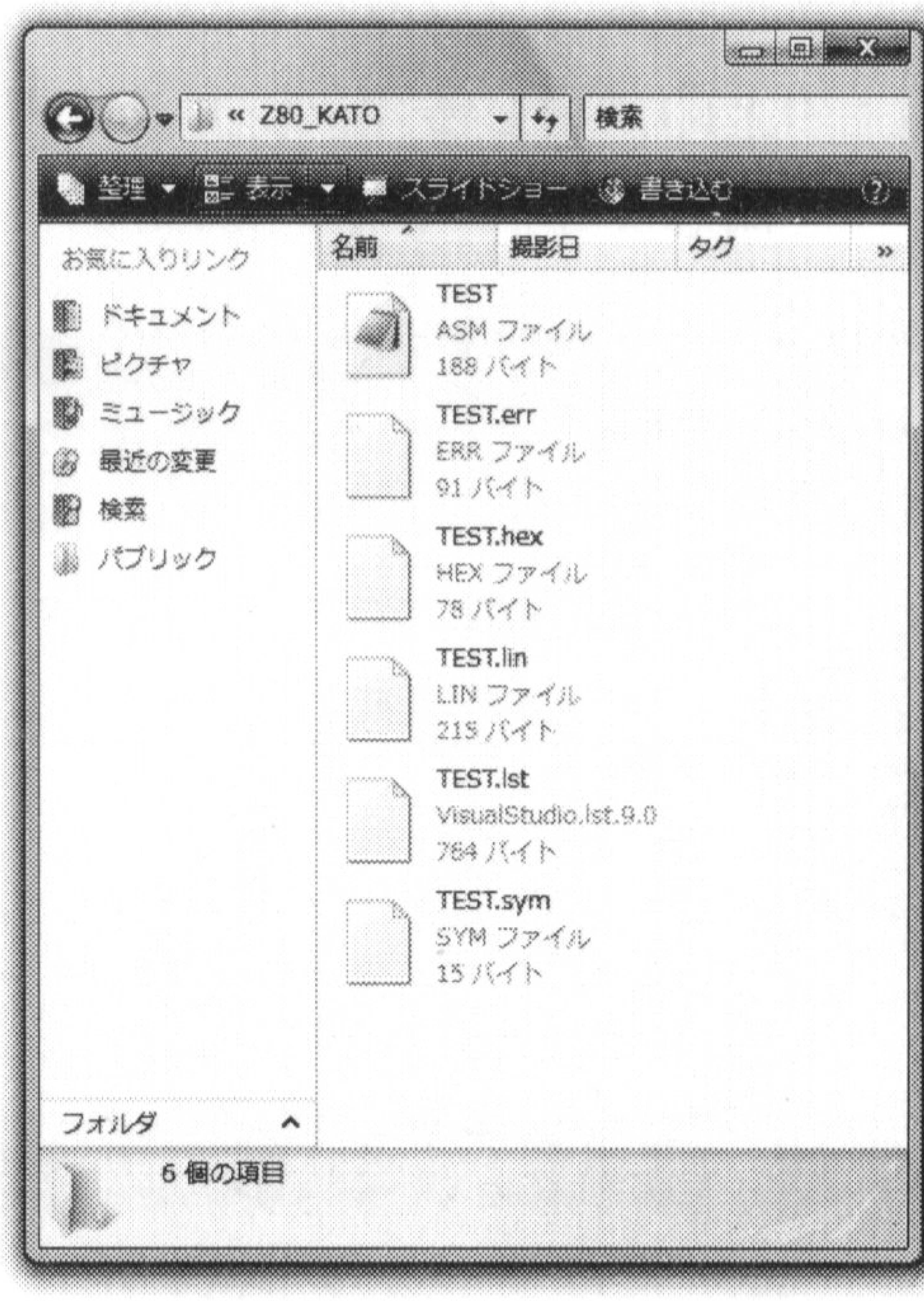


図3-14　1つのソース・ファイルから5つ生成される

まず、ヘキサファイル（HEXファイル）については説明不要でしょう。また、LINファイルとSYMファイルは、Z-Visionの表示画面をつくるためのファイルですので、閲覧することはほとんどありません。

ERRファイルは、XA80Wでアセンブルしたときの「警告&エラー情報」をそのまま記録したものです。図3-12の表示項目が多くて、内容を見落としたときなどに有効です。

もっとも役に立つのはリストファイル（LSTファイル）です。リスティングファイルとも呼ばれます。

リストファイルでは、最初に入力したアセンブリ言語によるプログラムリストや、アドレスと併記された機械語、さらにエラーメッセージ（1文字のアルファベットの記号で、何らかの記述のミスを知らせる）を一度にまとめて見ることができます。デバッグ作業に便利です。

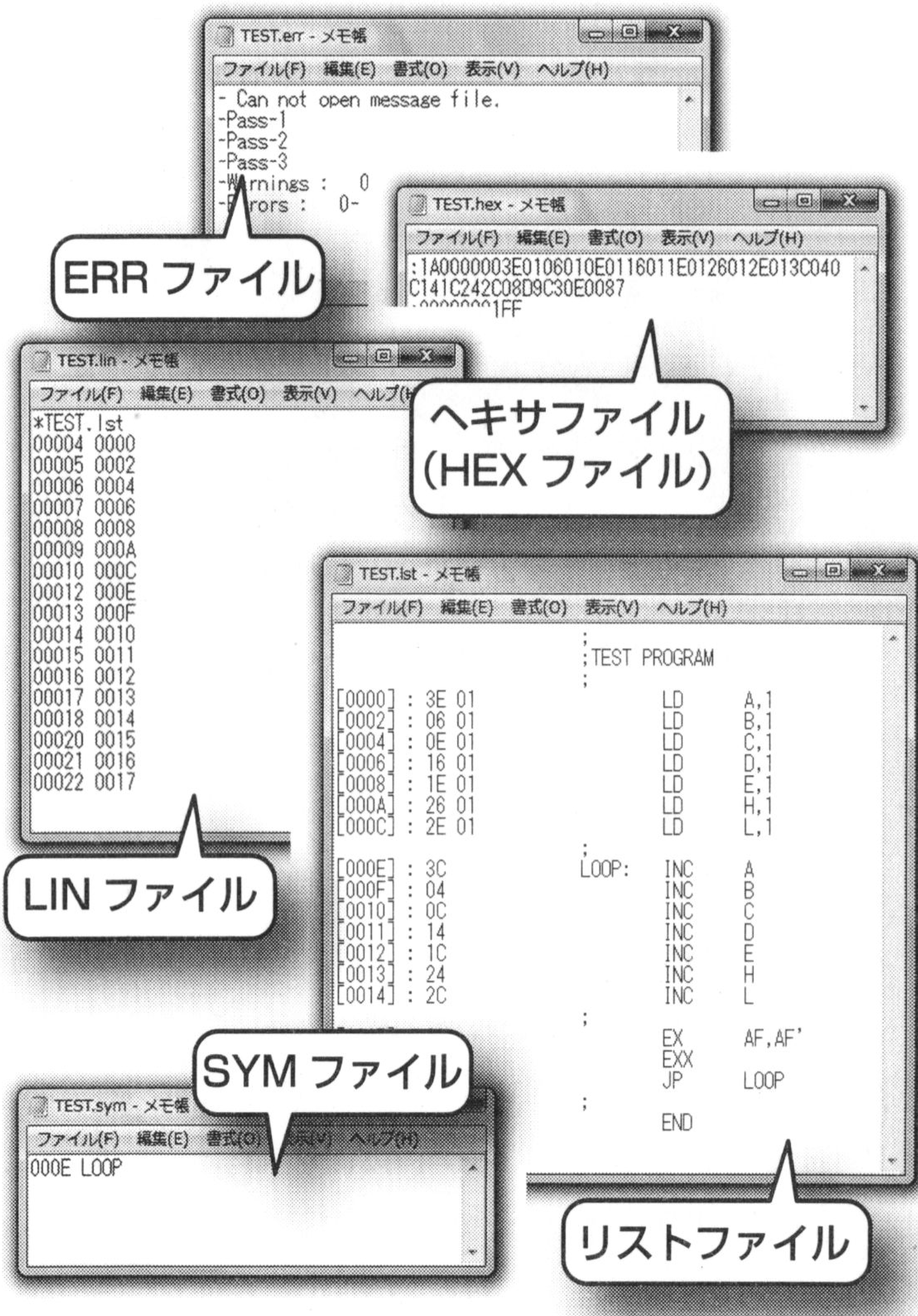


図3-15　5つのファイルの中身

3.5 テストプログラムをロードする

テストプログラムの準備がととのいましたので、機械語に変換された「TEST.HEX」を、シミュレータへロードしましょう。

①Z-Visionのウィンドウで、「LOAD」ボタンをクリックするか、ファイルメニューから「ファイルのロード」を選びます。

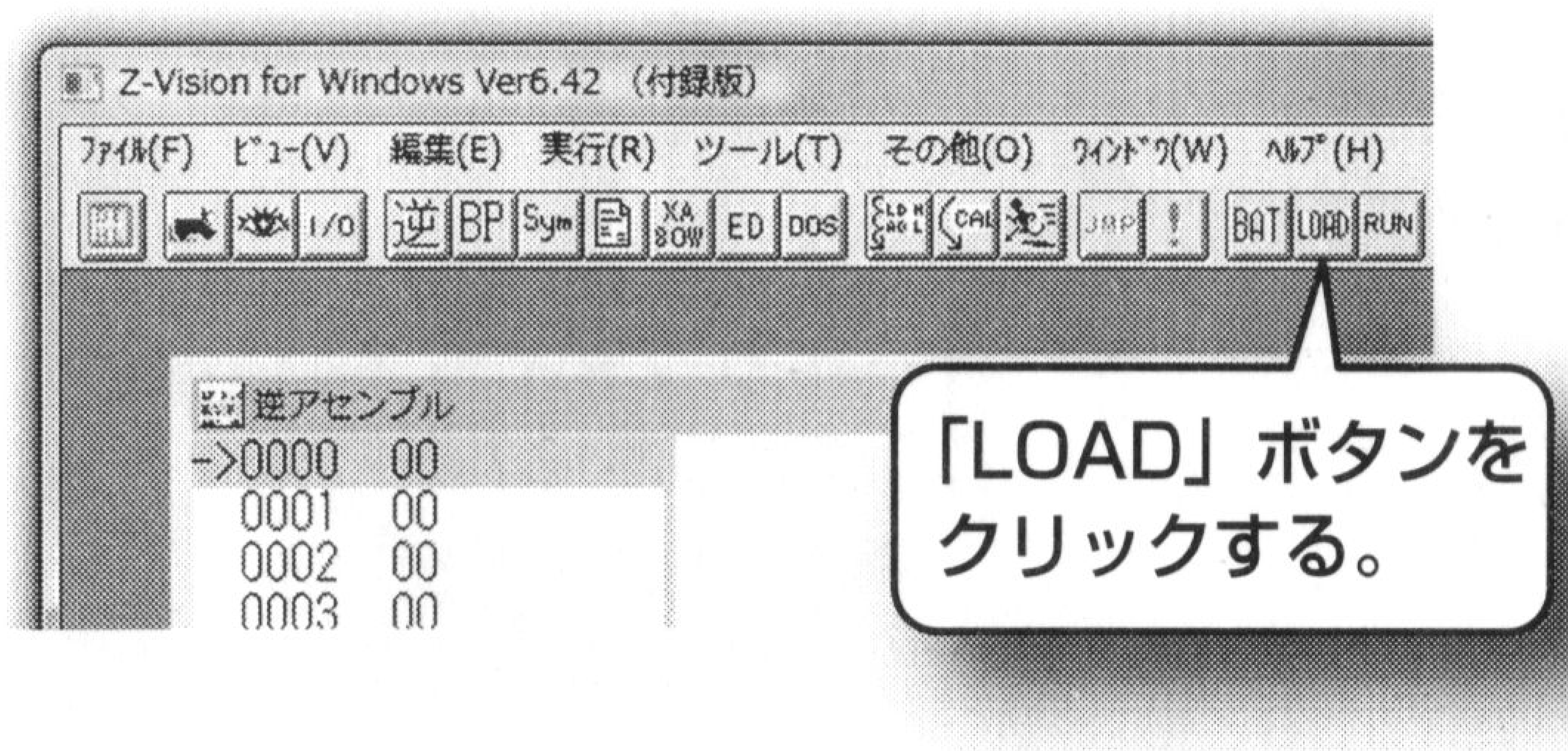


図3-16　ファイルをロードする

②「ファイルの読み込み」ダイアログボックスが開きます。ここでは、まず中央の「ディレクトリ」スクロールウィンドウで、保存用フォルダの名称が現れるように設定します。最初はとまどう方がおられますが、フォルダアイコンの操作方法が理解できれば、難しくありません。

図3-18では、Cドライブの中に保存用フォルダ（ここでは「Z80_KATO」）が見えていますので、このアイコンをダブルクリックします。なお、右下のチェックボックス☑には触れないでください。

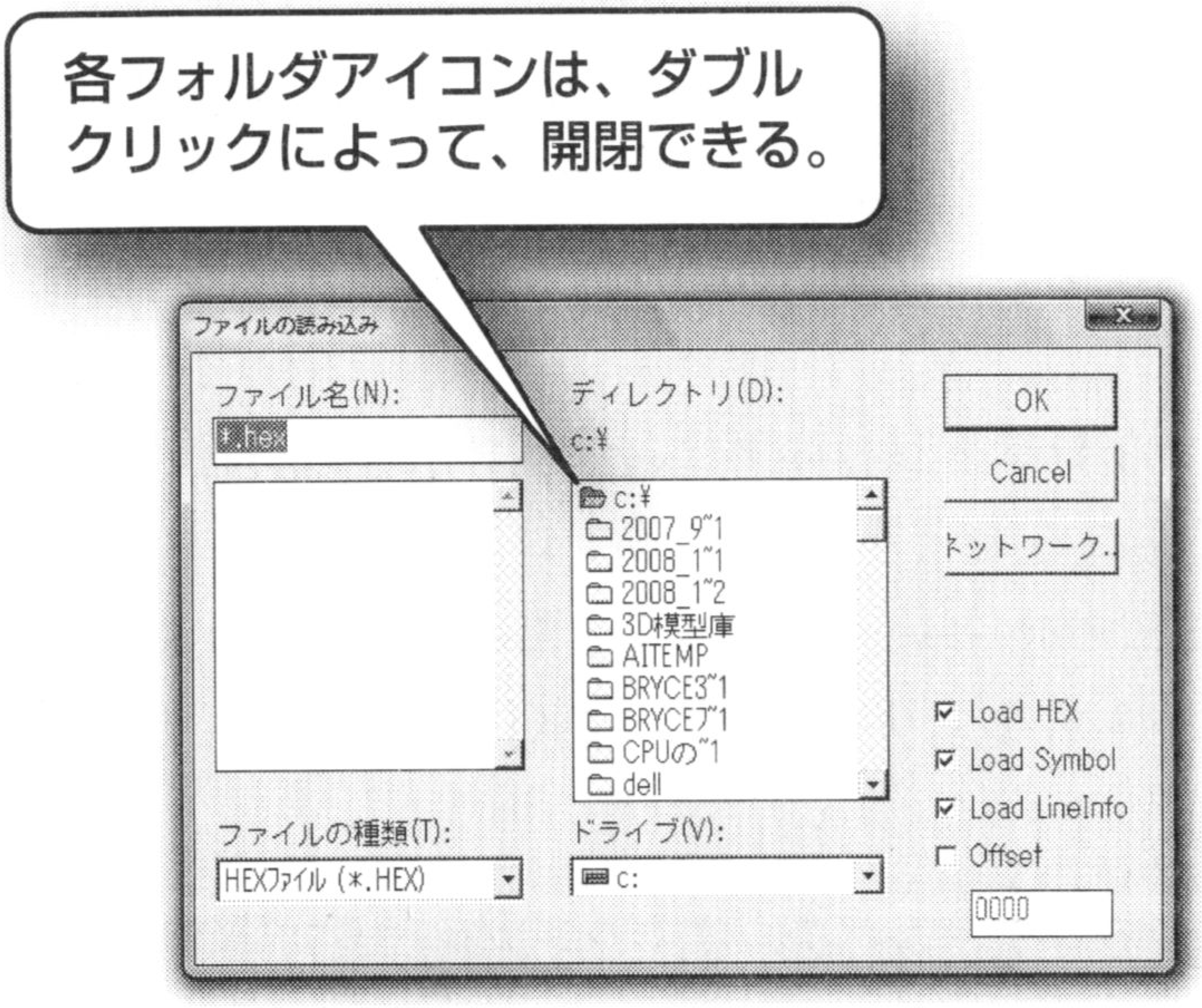


図3-17　保存用フォルダを探す

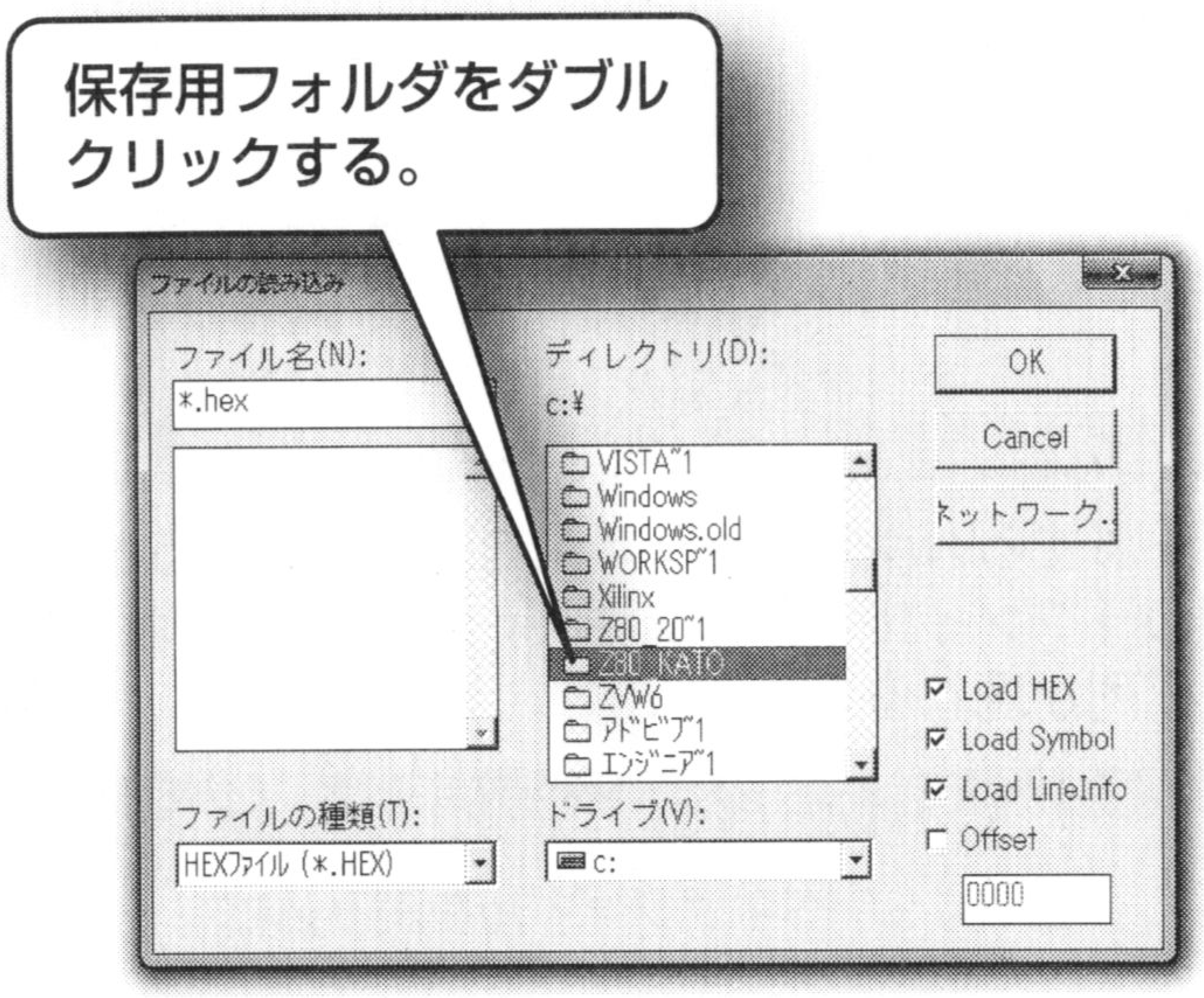


図3-18　保存用フォルダを開く

もし、すでにCドライブの別のフォルダを選んでいて、保存用フォルダが見えない場合は、スクロールウィンドウ内の「c:¥」のフォルダアイコンをダブルクリックすると、ディレクトリが1つ上がります。

③「ディレクトリ」の下に、保存フォルダが開いたアイコンが現れるとともに、ダイアログボックス左の空欄に「TEST.hex」というファイル名だけが表示されます。このファイル名をクリックして「OK」ボタンをクリックします。

繰り返しますが、拡張子に大文字/小文字の区別はありません。

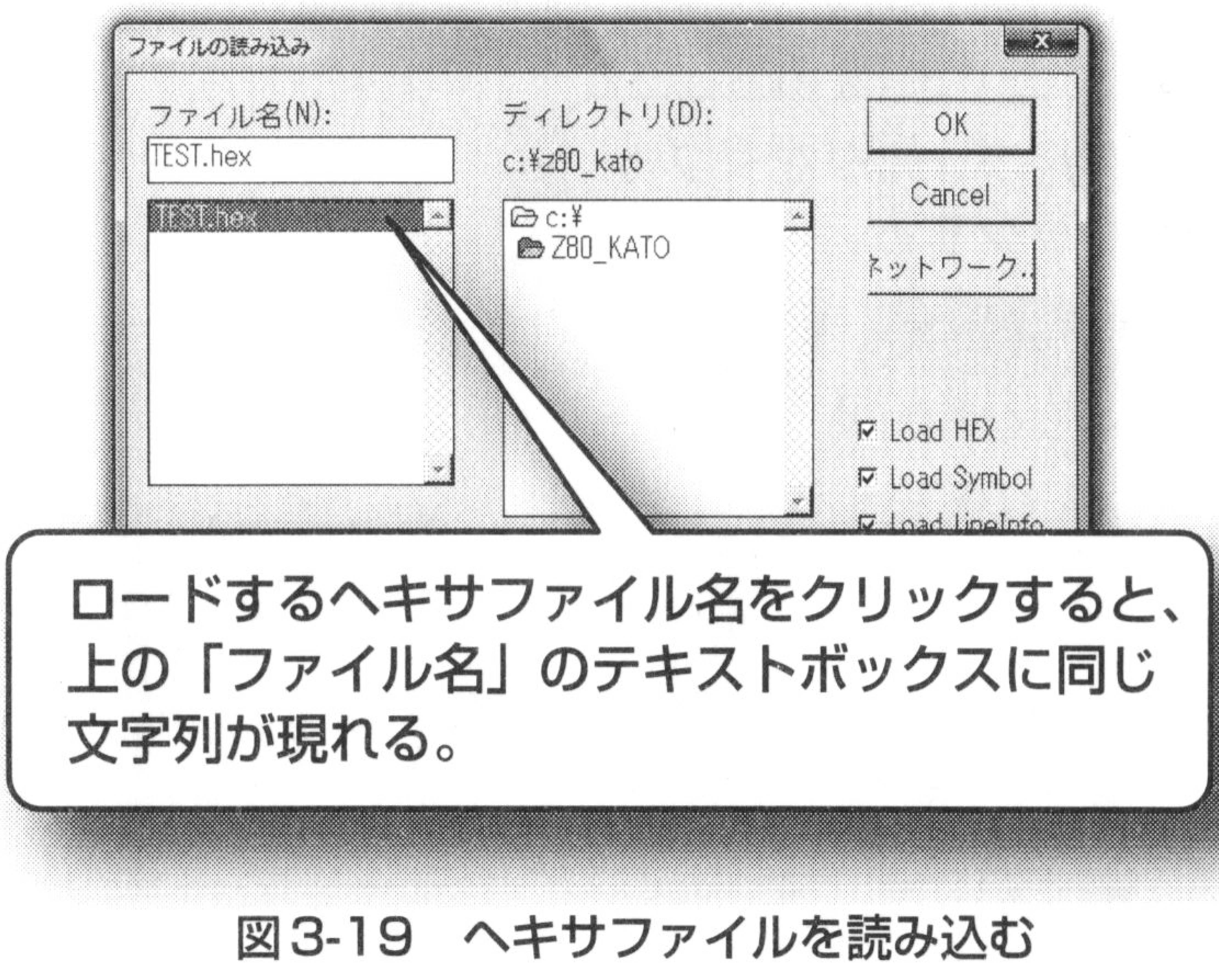


図3-19　ヘキサファイルを読み込む

ところで、今後、複数のヘキサファイルが生成されると、図3-19の「TEST.hex」という表示の下に、それらの

ファイル名が追加されることになります。

④ファイルのロードが始まります。「読み込みは終了しました」と表示されたら、「OK」ボタンをクリックします。



図3-20　読み込み完了

⑤逆アセンブルウィンドウとダンプウィンドウに、00以外の数字が追加されているのを確認することができます。これらは、すべて16進数です。

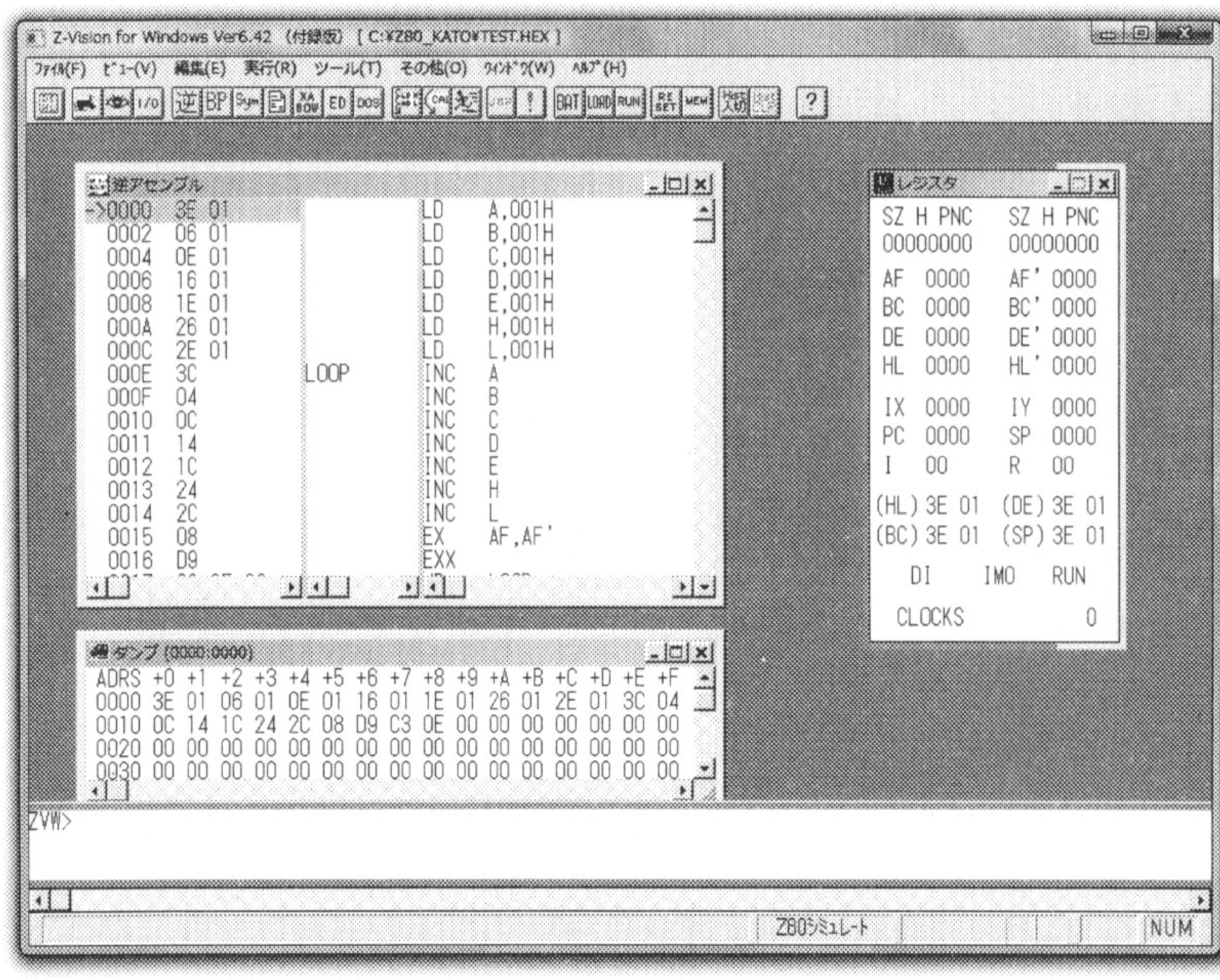


図3-21　テストプログラムがロードされた

第４章　シミュレータでCPUを見る

Z-VisionというCPUシミュレータが、Z80のどの部分を見せてくれるのかが、この章のテーマです。

一般にCPUは、ディジタルICの中でもとりわけ複雑な回路システムです。多くのブロックに分かれて、それぞれが重要な働きをしていますが、すべてをパソコン画面に映し出すことは困難ですし、見えてはいてもその情報が役に立たないのであれば、意味がありません。

しかし、Z-VisionはZ80の必要十分なところだけを見せてくれますし、それ以上の情報は必要ないと言ってよいと思います。

その理由は、シミュレータを含んだデバッガが、プログラミングをいかに短時間で効率的に行うかに目的を置いて開発されているからです。

ここでは、第３章で入力したテストプログラムをZ-Visionにロードして、実際のCPUの中身と比べてみましょう。

4.1　コンピュータの構造

Z-Visionにロードされたプログラム情報は、関連するウィンドウに反映されますが、これを理解するためには、実際のZ80マイコンシステムの知識が少し必要です。

これまで、コンピュータという言葉をきちんと定義していなかったのですが、本書では、データを記憶するメモリやI/Oポートなどを含めて述べるときにはコンピュータ、中央演算処理装置であることを特に強調するときはCPUとして表記しています。

たとえば、コンピュータを学ぶときには、必ずと言ってよいほど、図4-1のような図が使われます。

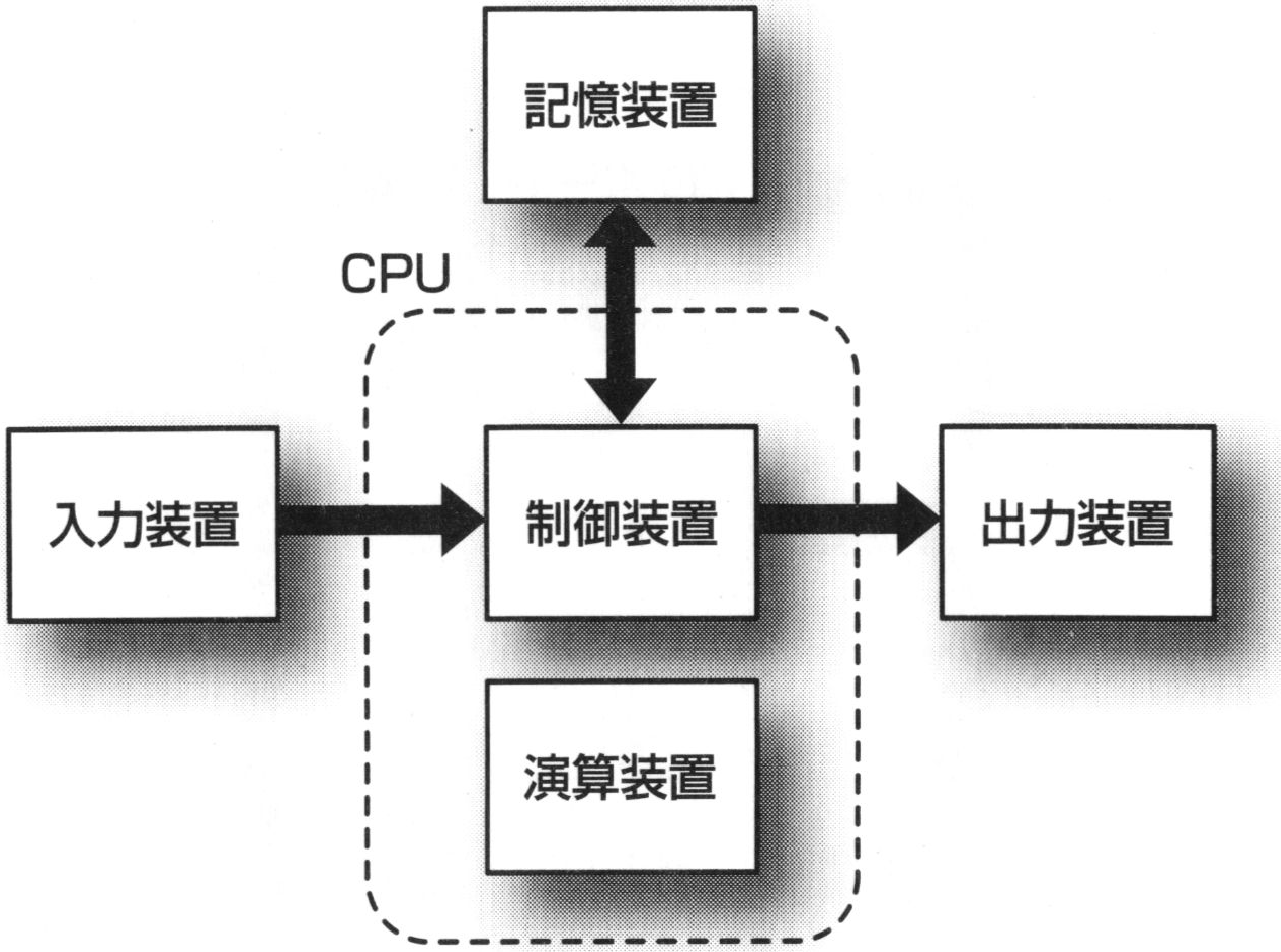


図4-1　コンピュータシステムの基本形

図4-1は、フォン・ノイマン型アーキテクチャと呼ばれるコンピュータシステムの基本形を、ブロック図で表したものです。ブロック図というのは、まとまった機能を持つ電子回路のかたまりを四角で表現したものです。

次に、このブロック図をZ80マイコンシステムに合うよう書き直したものが、図4-2です。

入力装置と出力装置は、対（つい）の機能として1つにまとめられていますので、これをI/Oポート（入出力ポートの意味）としました。また、記憶装置はメモリとし、各ブロックをつないでいる信号線の束の名称も、明確に記入しました。

この図をもとにして、話を進めましょう。

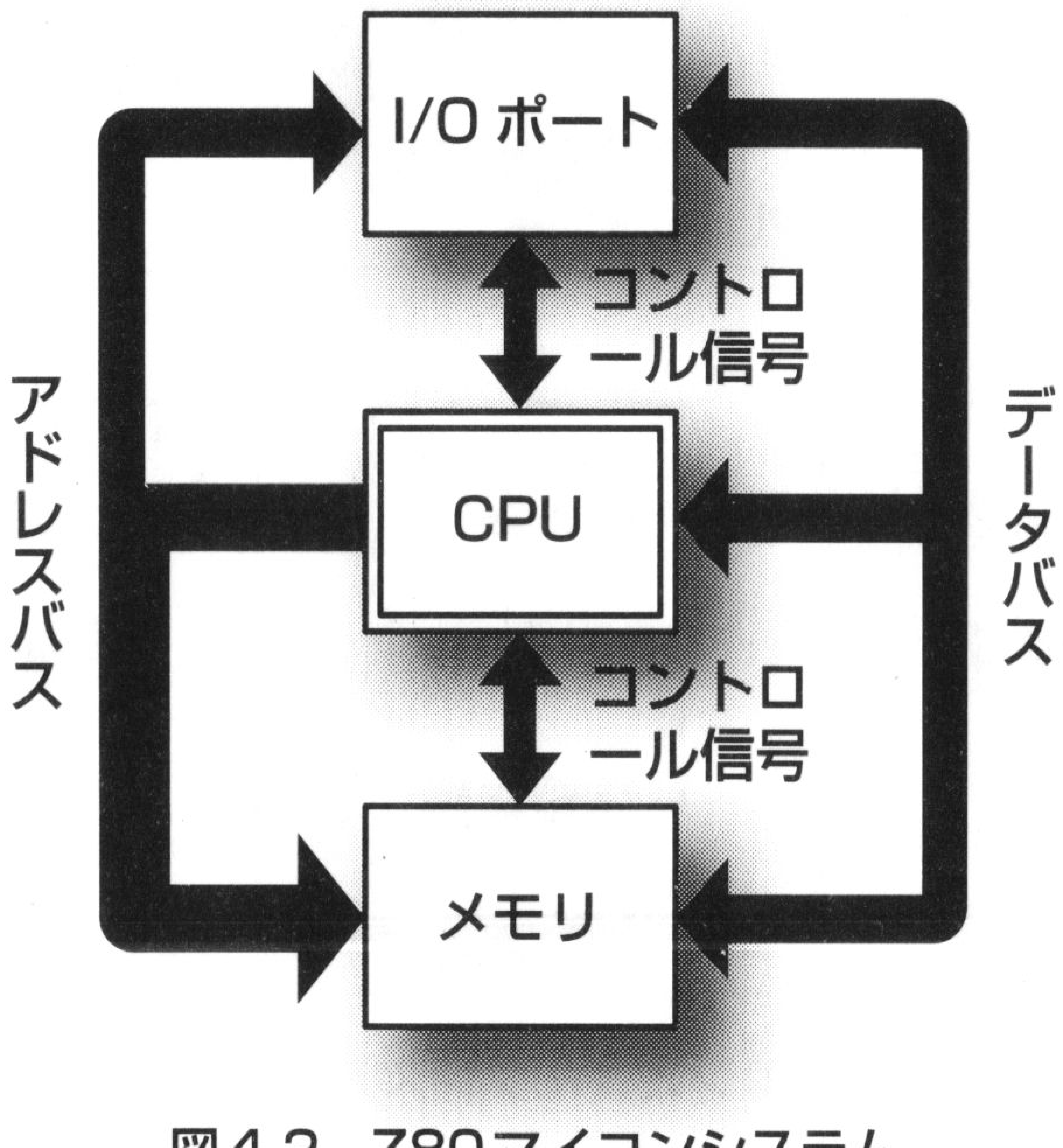


図4-2　Z80マイコンシステム

4.2　メモリのアドレスとデータ

図4-2には、アドレスバスとデータバスという黒い帯があります。

アドレス（address）とは、文字通り住所のことです。コンピュータでは、CPUにつながっているメモリ、つまりデータの記憶装置に住所を割り当てており、それは単なる数字の羅列です。

私たちが生活している住所のイメージで言うなら、たとえば「1234番地」ということになります。

また、第7章でシミュレートしますが、I/Oポートも同様にアドレスで管理されています。ただし、メモリとI/Oポートが同じアドレスだと区別できないため、どちらを利用するのかについては、命令語の種類を変えて、そのときのコントロール信号によって、メモリとI/Oポートが同時に動作しないようにできているのです。

なお「バス」は、町を走る乗り合いバスをイメージして付けられた言葉で、アドレスバスはアドレス信号線の束、データバスはデータ信号線の束を指します。1台のバスに16人が乗る様子が、16ビットのデータが同時に運ばれるイメージです。

Z-Visionを見ながら、詳しく見ていきましょう。

図4-3は、Z80マイコンシステムと逆アセンブルウィンドウを並べたものです。実は、メモリの内容が逆アセンブルウィンドウに表示されています。

逆アセンブルというのは、アセンブル、つまりアセンブリ言語から機械語への翻訳とは逆の操作をすることです。

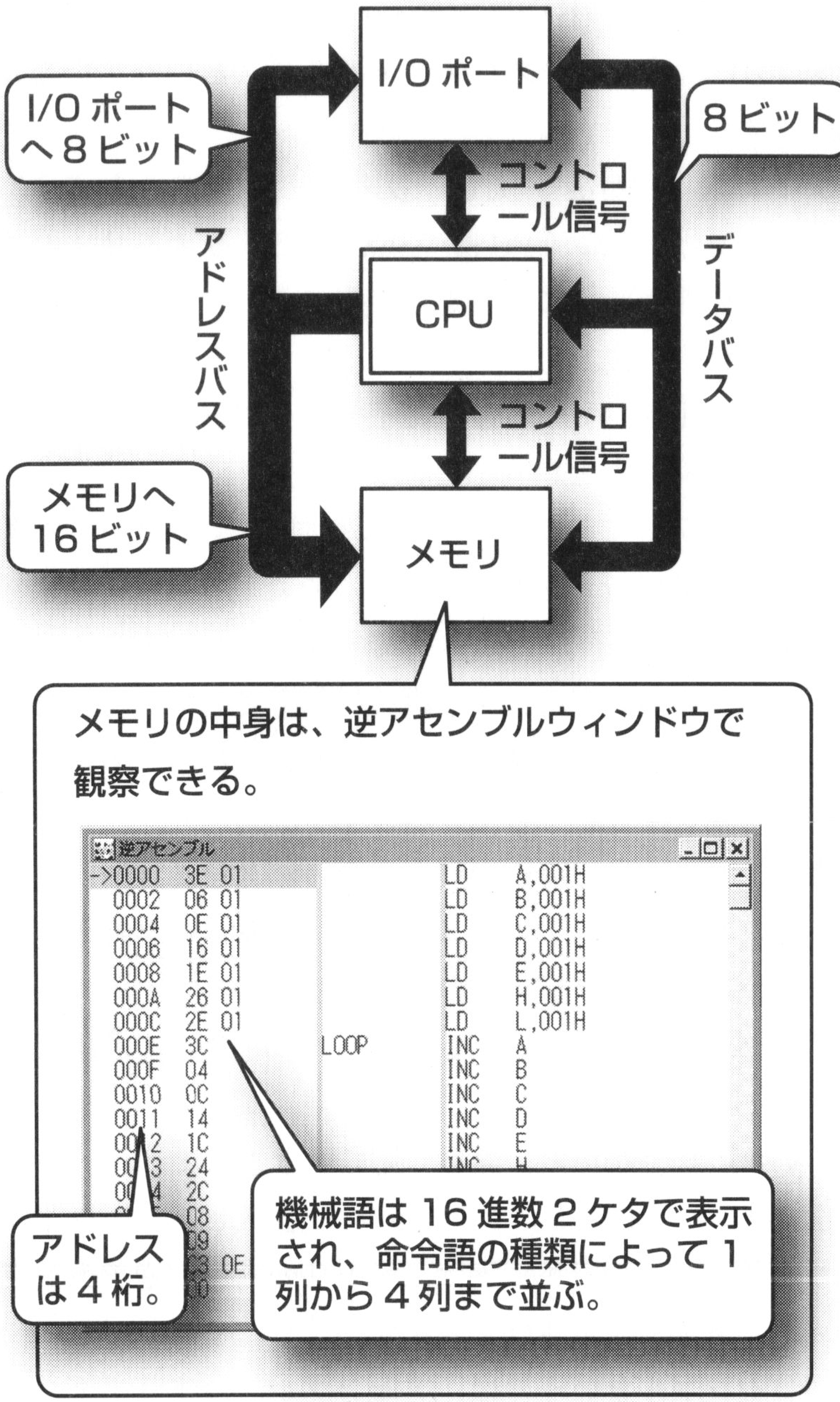


図4-3　メモリと逆アセンブルウィンドウ

逆アセンブルウィンドウでは、XA80Wで生成された各種ファイルの情報を連携して、プログラムリストを再現しているのです。アセンブリ言語で記述した命令文の左側に、メモリのアドレスと機械語が表示されます。

この逆アセンブル作業は、Z-Visionが自動的に行っているため、第3章で作成した図3-2のリストとは少し違っています。たとえば、コメント文は削除されていますし、数値の1は001Hに置き換わっています。ENDという命令文もありません。

しかし、Z80が実行すべき命令文の意味が変更されたわけではなく、見かけの表現が変わっただけです。



図4-4　最初のプログラムリスト

逆アセンブルウィンドウの表示について補足しましょう。

Z80の場合、２進数で16桁をアドレスとしてメモリに割り当てており、16進数で表現すると、0000H番地からFFFFH番地ということになります。ちなみに、２進数で16桁のことを16ビットと呼びます。

アドレスは物を数えた値ではありませんので、ゼロの部分も番地の一部です。したがって、本来0000H番地は、数字の「０」１つだけで表記すべきではありません。

確かに0000を０に省略するのは、数値の扱い方としてごく普通ですが、アドレスであることがはっきりとわかるよう、４桁のまま表記することが望ましいと考えます。



図4-5　１つ１つのメモリはアドレスで指定する

4.3　CPUの中のメモリ

コンピュータは電子計算機と訳されるように、計算することが最大の機能です。しかし、直接的に計算する部分が効率的に動くためには、どうしても一時的に数値を記憶するメモリが必要です。

それが、CPUに内蔵されているレジスタです。

図4-6は、Z80の内部を示すものですが、レジスタは１つではありません。ブロック図には命令レジスタと各種レジスタの２つがあります。

実は、私たちユーザがプログラムで利用できるのは各種レジスタで、Z-Visionにも表示されます。これについてはあとで述べるとして、ここでは、CPUが何をしているのかをおおまかに理解しましょう。

以下は、CPUが１バイト命令（6.1節参照）を実行する手順です。図4-6を見ながら、ゆっくりお読みください。

①まずCPUは、これから実行する命令語が格納されているアドレス（住所の番地）を、アドレスバス経由でCPU外部につながっているメモリに連絡します。

②メモリには多くの部屋があって、それぞれ異なるアドレスを持っています。CPUが指定したアドレスに該当するメモリの部屋には、１バイトのデータが入っていますので、この値が読み出されて、今度はデータバス経由でCPUに入ってきます。

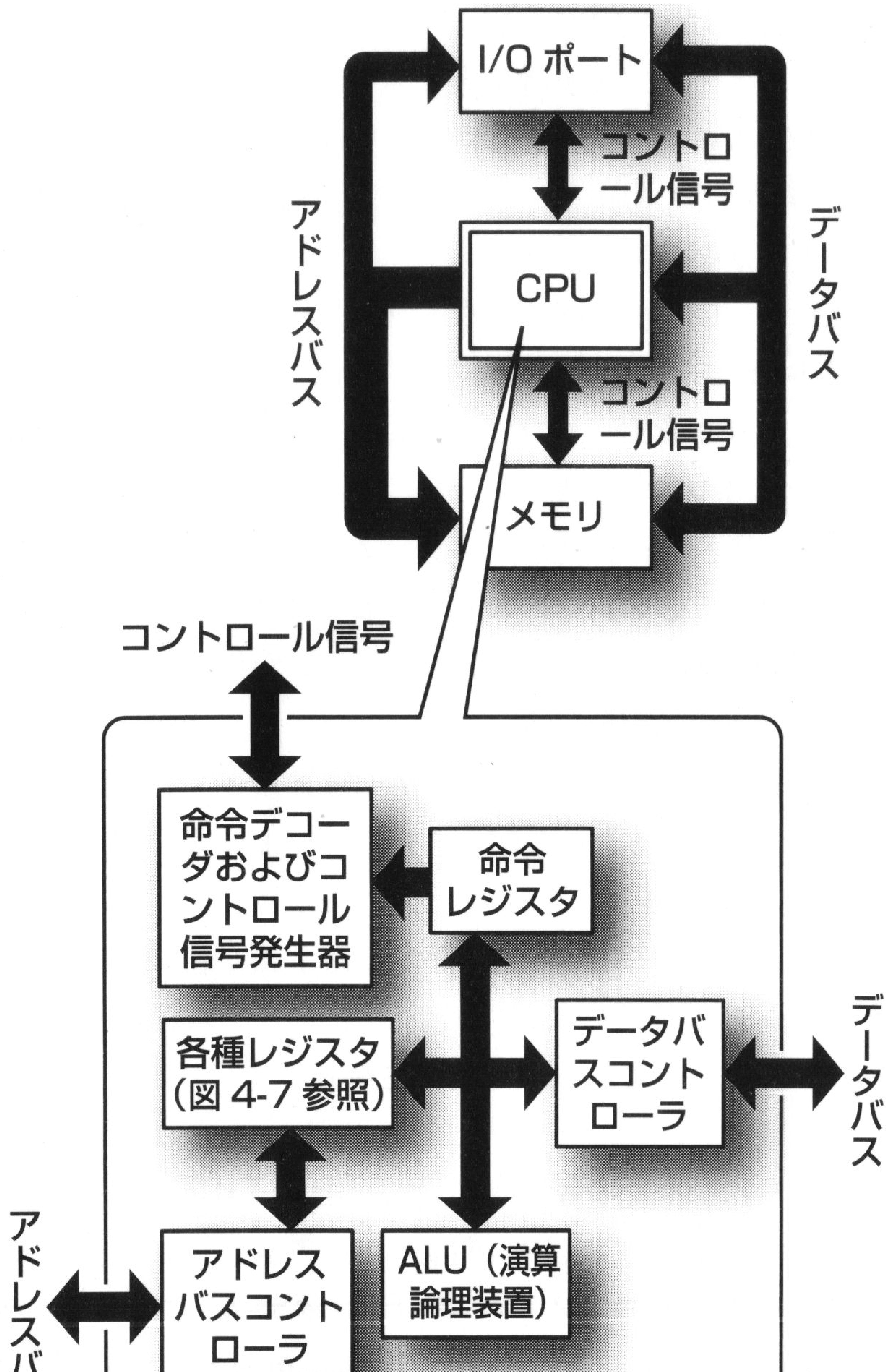

I/O ポート
コントロール信号
アドレスバス
データバス
CPU
コントロール信号
メモリ
コントロール信号
命令デコーダおよびコントロール信号発生器
命令レジスタ
各種レジスタ（図 4-7 参照）
データバスコントローラ
データバス
アドレスバス
アドレスバスコントローラ
ALU（演算論理装置）

図4-6　Z80の内部

③データバスを通って入ってきたメモリの内容は、命令語です。この命令語は、CPU内部のデータバスを通って、命令レジスタに入り、一時記憶されます。

④その後、命令語は**命令デコーダ**の中で解読が行われ、その結果をもとに、その命令を実行するために必要な各種の**コントロール信号**が作られます。

⑤また、命令の中には、**算術演算**（加算、減算など）や**論理演算**（AND、ORなど）の演算をするものがあり、その場合は**ALU**（arithmetic logic unit：**演算論理装置**）にデータが送られます。

4.4　レジスタの種類

CPUのレジスタは種類が多く、それぞれの略語がアセンブリ言語のニーモニックとして表現されています。そのため、本来ならば、プログラムをつくりながら１つ１つ理解するのが望ましいと考えます。

しかし、レジスタ全体から見た、個々のレジスタの位置づけも重要ですので、なるべく手短にすべてのレジスタについて紹介していきます。

なお、図4-7は、実際のZ80のレジスタ群、そして図4-8はZ-Visionで表示されるレジスタウィンドウの様子です。これら２つの図を比べながらお読みください。

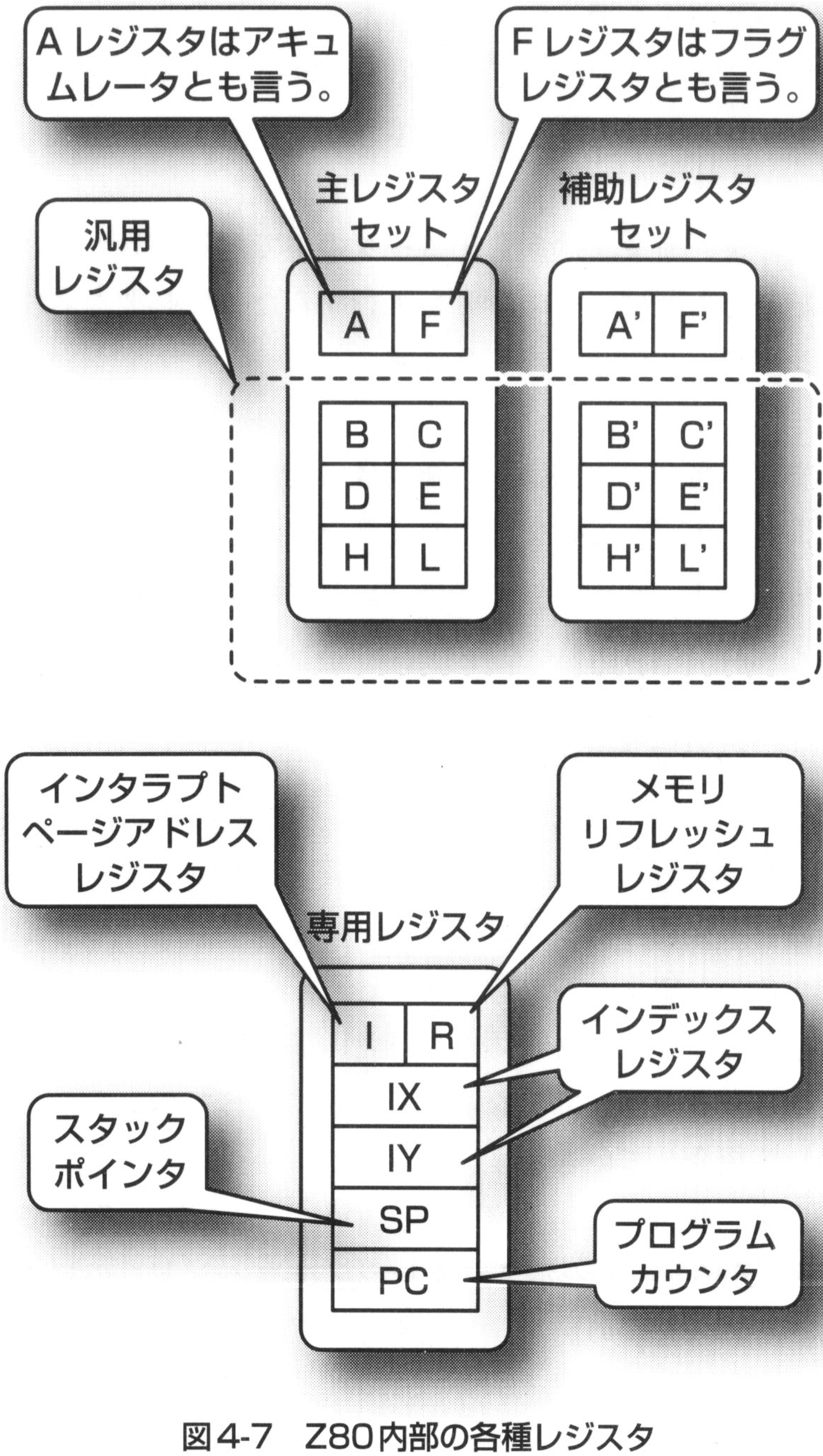


図4-7　Z80内部の各種レジスタ

①全体は3グループ

Z80のレジスタは、大きく3つのグループに分かれています。主レジスタセットと補助レジスタセット、専用レジスタです。

主レジスタセットと補助レジスタセットは、ちょうど背中合わせに存在するレジスタ群で、両方を同時に使うことはできません。

詳しくは、あとの④汎用レジスタで述べますが、A、F、B、C、D、E、H、Lの計8つのレジスタと、それぞれにクォーテーションマーク（'）を付けた補助レジスタセットは、それぞれ独立しています。いずれも8ビットのデータを記憶できるレジスタですが、必要に応じて「表と裏」、たとえば「AとA'」の値を入れ替えて利用する仕組みです。

ちなみに、16ビットの値を扱う場合、「BC」が表なら「BC'」と表します。

3つ目の専用レジスタは、CPUに不可欠なアドレスの管理や、特殊な処理をするためのレジスタ群です。

②Aレジスタ（8ビット）

もっとも忙しく、中心的に働きます。アルファベットのAは（accumulator：アキュムレータ）という正式名称の頭文字です。アキュムレータは、accumulate（蓄積する）という言葉から名付けられました。つまり、もともとALUと連携して、算術演算や論理演算を行うために用意されたレジスタです。

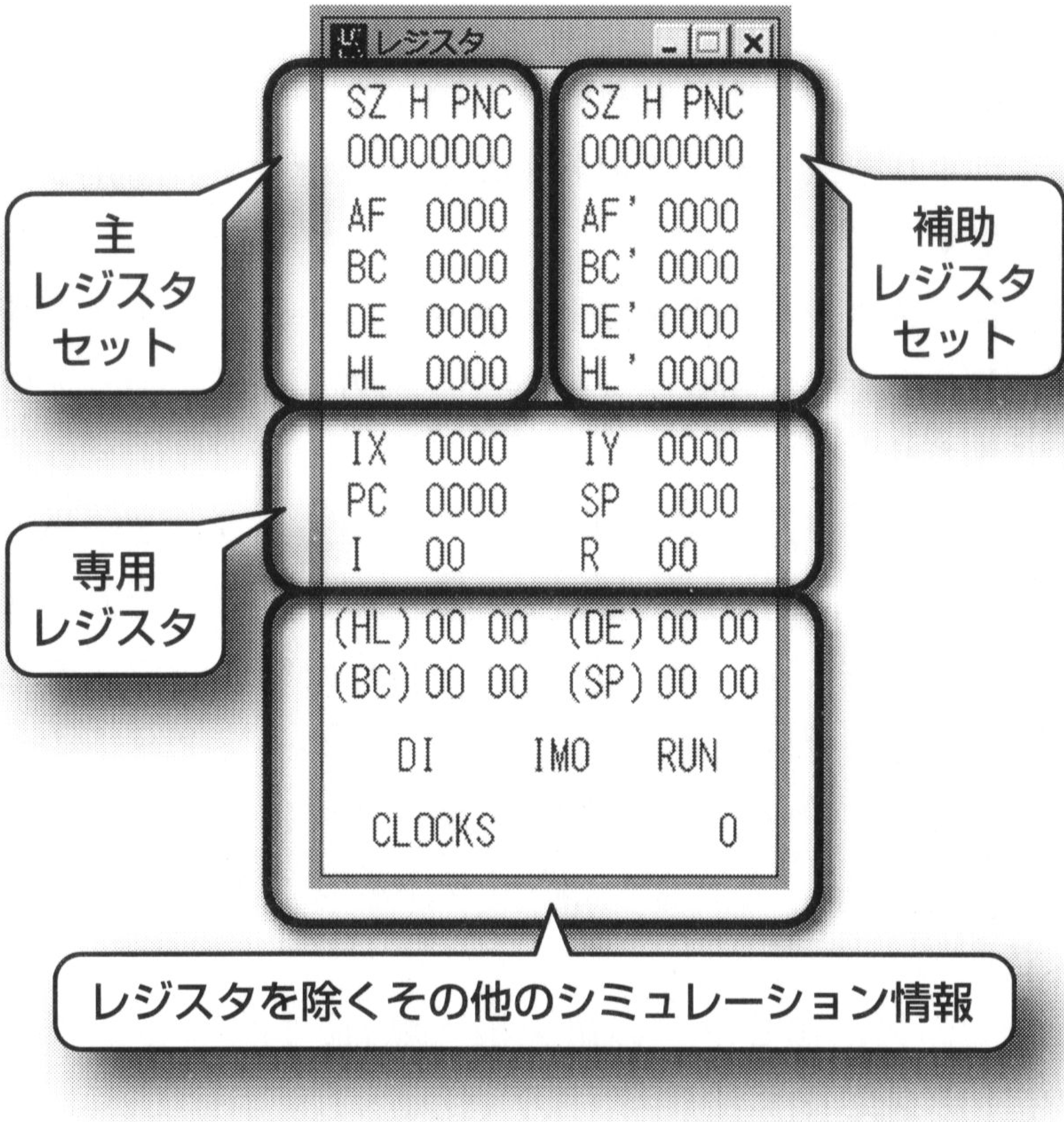


図4-8　Z-Visionのレジスタウィンドウ

しかし、実際のAレジスタは、Z80の命令語の大半にかかわることができます。たとえば、

■ビットシフト（2進数の各桁を左右に移動すること）
■すべてのレジスタやメモリの間のデータ移動
■CPU外部とのデータの入出力

などです。

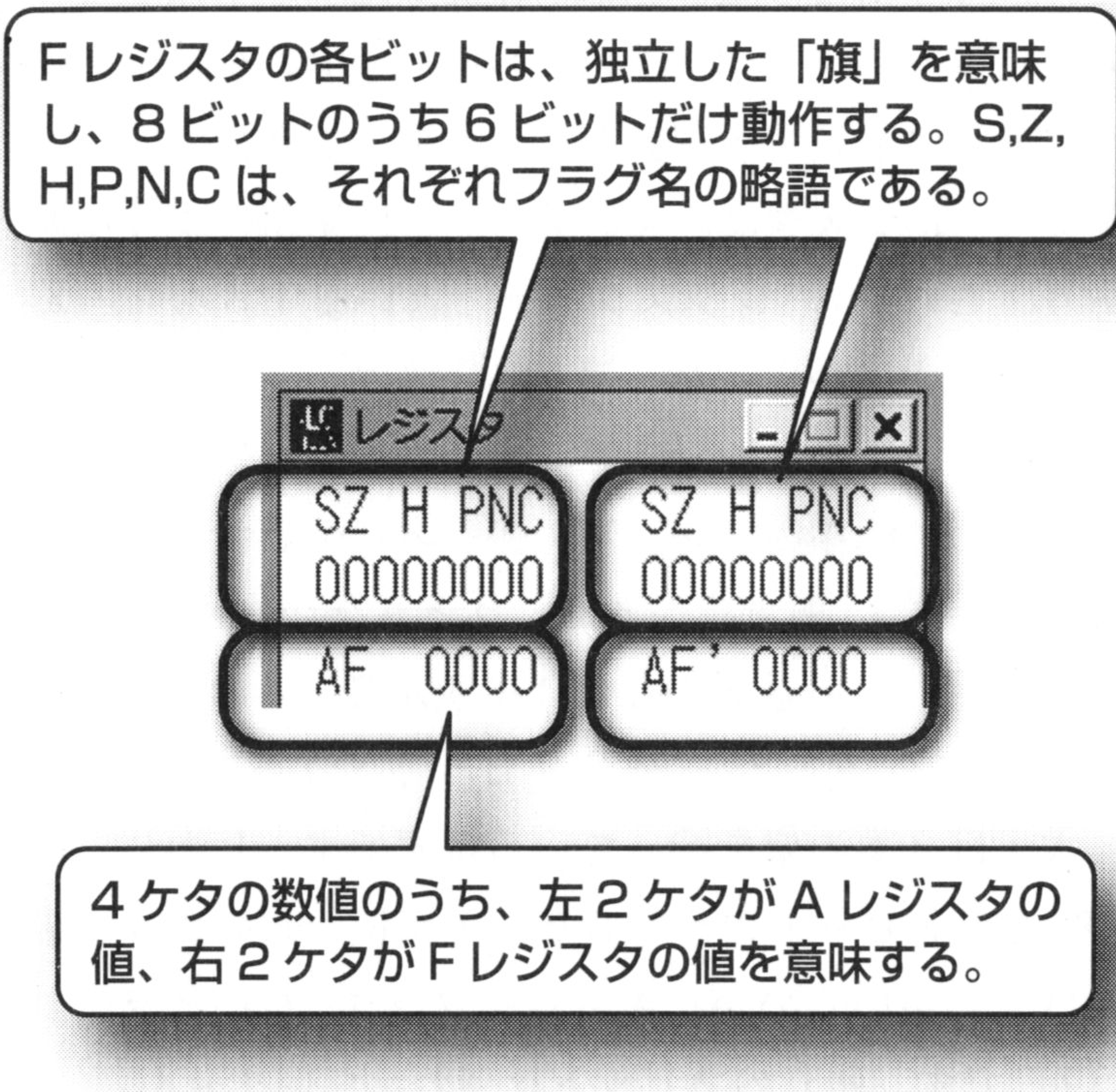


図4-9　AレジスタとFレジスタ

③Fレジスタ（8ビット）

アルファベットのFは、フラグ（flag：旗）の頭文字をとったもので、Aレジスタの演算結果に応じて、独立した各ビットの値を１（旗を揚げる）にしたり０（旗を降ろす）にしたりする働きがあります。

主に、条件付きジャンプ命令が機能するための条件を与える役割をしています。

条件付きジャンプ命令というのは、Aレジスタの演算結果がたとえば「ゼロになったら」とか「プラスだったら」

という条件を満たしたときに、指定したアドレス（メモリ上の住所）へジャンプ（CPUが次に実行する命令が格納されているアドレスへ移動）する命令のことです。

つまりFレジスタは、Aレジスタとは兄弟のような関係にあり、これをAFレジスタペアと言います。もちろんこれら2つは機能がまったく異なりますが、Aレジスタの仕事の結果が、直接Fレジスタに現れる仕組みです。

そのため、図4-8のレジスタウィンドウでも、AF 0000と表示させています。これは、AとFをつなぐと16ビットであるため、16進数の0が4桁並んでいるのです。

ところで、Fレジスタは8ビットのレジスタですから、本来は8本の旗を持っているはずです。しかし、実際のZ80では8ビットのうち6ビットが使われており、最もひんぱんに用いるのが、キャリーフラグ（C）とゼロフラグ（Z）です。残りの2ビットは使われていません。

この様子を、Z-Visionのレジスタウィンドウでは、最も上の部分に表しています（図4-9参照）。アルファベットのS、Z、H、P、N、Cが、それぞれのフラグの頭文字です。これらは、条件付きジャンプ命令と直接かかわっていますので、7.9節で詳しく述べます。

キャリーフラグの略語には「C」と「CY」の2種類があって、両方とも使われています。本書の説明では「CY」を使いますが、Z-Visionの中ではCレジスタのCもキャリーフラグのCも同じ表現ですので、注意が必要です。

④汎用レジスタ（各8ビット）

主レジスタと補助レジスタのうち、A、F、A'、F'の4つを除くレジスタを汎用レジスタと呼びます。「汎用」ですから、自由に8ビットレジスタとして使えます。ただし、Aレジスタとは違って使用範囲が狭いのも特徴です。

先にも述べましたが、主レジスタセットの中の汎用レジスタは、B、C、D、E、H、Lの6つのレジスタで構成されており、補助レジスタセットとは、「表と裏」の関係です。

言いかえると、「主」と「補助」は、レジスタごとに背中合わせに配置されていて、プログラマからはどちらか一方の「値」しか扱うことができません。

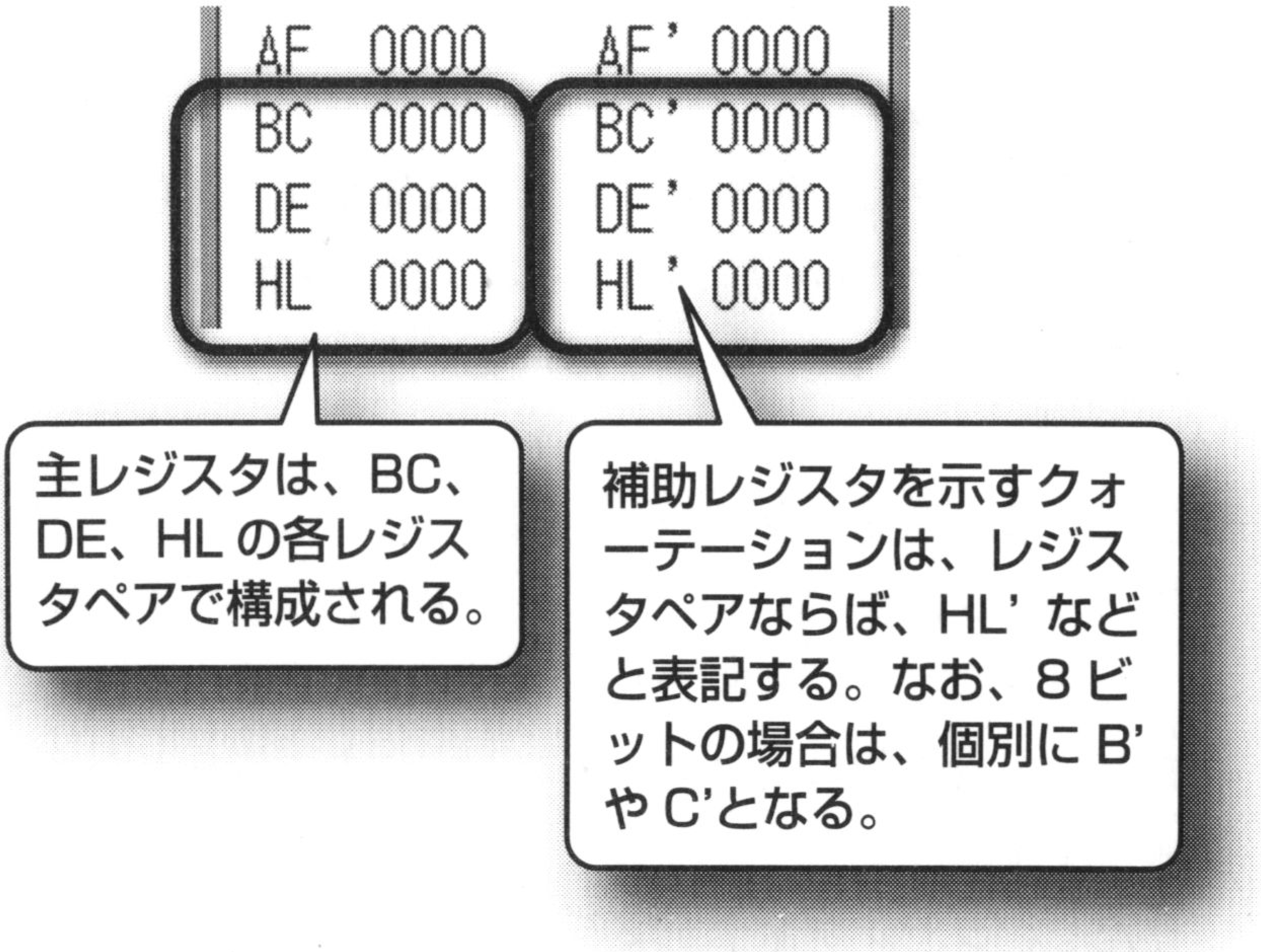


図4-10　汎用レジスタ

必要に応じて「表の顔」と「裏の顔」を入れ替えるのです。実は、入力していただいたテストプログラムの中には、この表裏を入れ替える命令語（EXとEXX）が入っています。

さて、これら汎用レジスタにはもう1つ特徴があります。レジスタペアと言って、BC、DE、HLを必要に応じて 8ビット＋8ビット＝16ビット 幅の数値を扱うことができ、これをレジスタペアと呼びます。裏側の補助レジスタセットも同様で、BC'、DE'、HL'と書きます。

つまり、先に述べたAFおよびAF'レジスタペアは特殊なもので、その他のレジスタペアと目的が異なることがわかります。

⑤専用レジスタ

専用レジスタは、汎用レジスタとは異なり、あらかじめCPU内部での役割が決まっているレジスタ群で、プログラムを実行する上でひじょうに重要です。

■プログラムカウンタ（16ビット）

PCと略します。16ビットのカウンタで、メモリのアドレスを指定します。一度アドレスバスにアドレスを送出したあとは、自動的にインクリメント、つまり値を1だけプラスします。

ただし、ジャンプ命令が実行されると、その指示にしたがって、離れたアドレス値に変更されます。

■スタックポインタ（16ビット）

SPと略します。16ビットのアドレス専用のレジスタで、CPU外部のメモリのアドレス値を持ちます。

たとえばサブルーチンと言って繰り返し使われるプログラムのブロックを、メインルーチンつまりプログラムの本体から離れたアドレスに配置して、必要に応じてサブルーチンを呼び出すときに、このSPが使われます。

7.10節で実際の動きを見ていただきますが、メインルーチンからサブルーチンを呼び出すと、PCはサブルーチンのアドレスを指定します。このままでは、メインルーチンに戻ってくることができません。そこで、メインルーチンに戻るときのアドレス「**戻り番地**」をSPに預けておくのです。

SPは、自分が管理しているメモリのスタックという記憶領域に、逐次「戻り番地」を保存していきます。この仕組みは、サブルーチン以外の目的でも使われます。

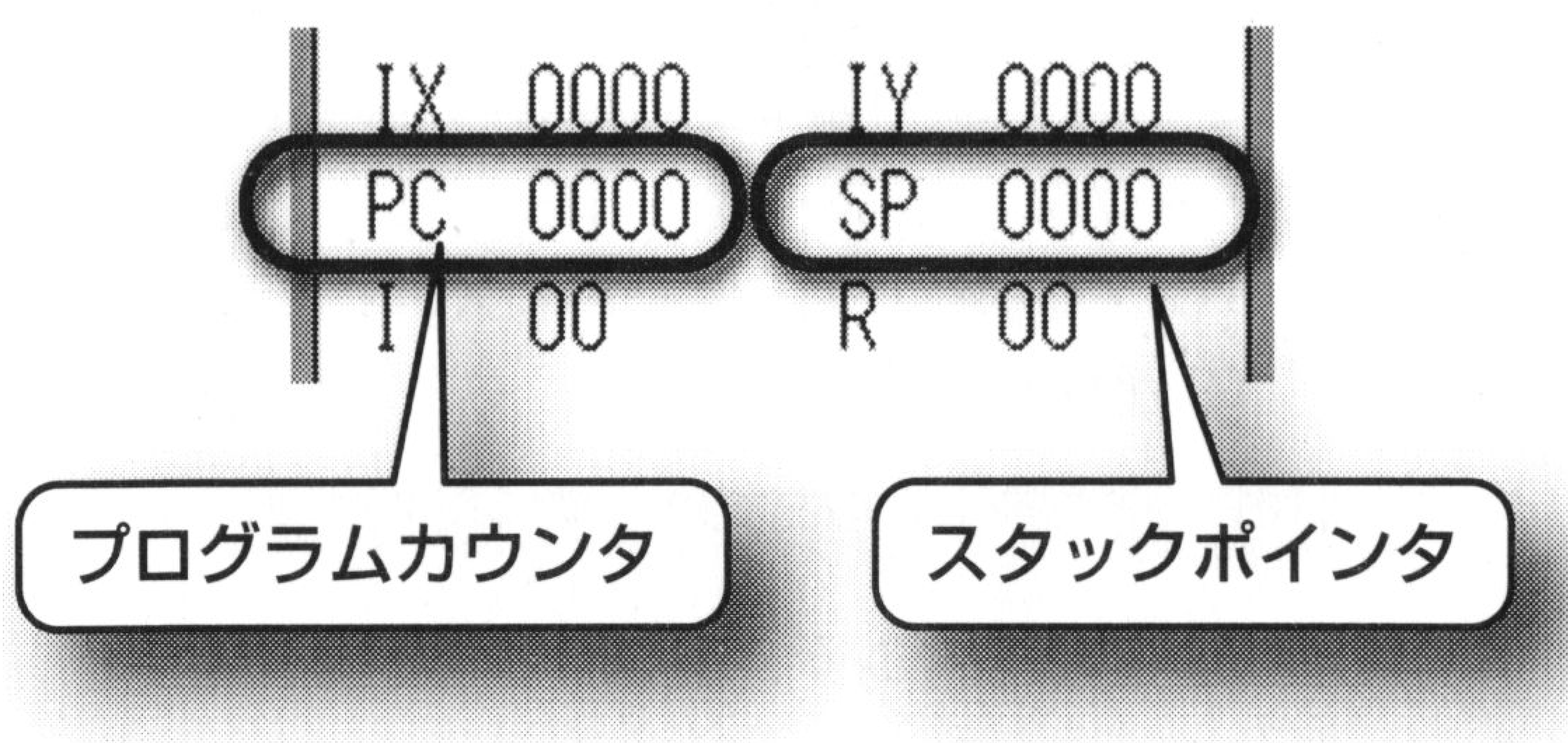


図4-11　PCとSP

■インデックスレジスタ（16ビット）

インデックスレジスタも、SPと同様に16ビットのアドレス専用のレジスタです。IXとIYの独立した2つがあります。

特有なのは、アドレス計算用の加算回路が付いているという点です。多数のデータ群をプログラム内に配置するような、これを一般にデータテーブルと呼びますが、こうした場面で威力を発揮する便利なものです。

しかし、ある程度プログラミングに慣れてきたときに使うのが望ましいと考えます。そのため本書では、これらのレジスタには触れていません。

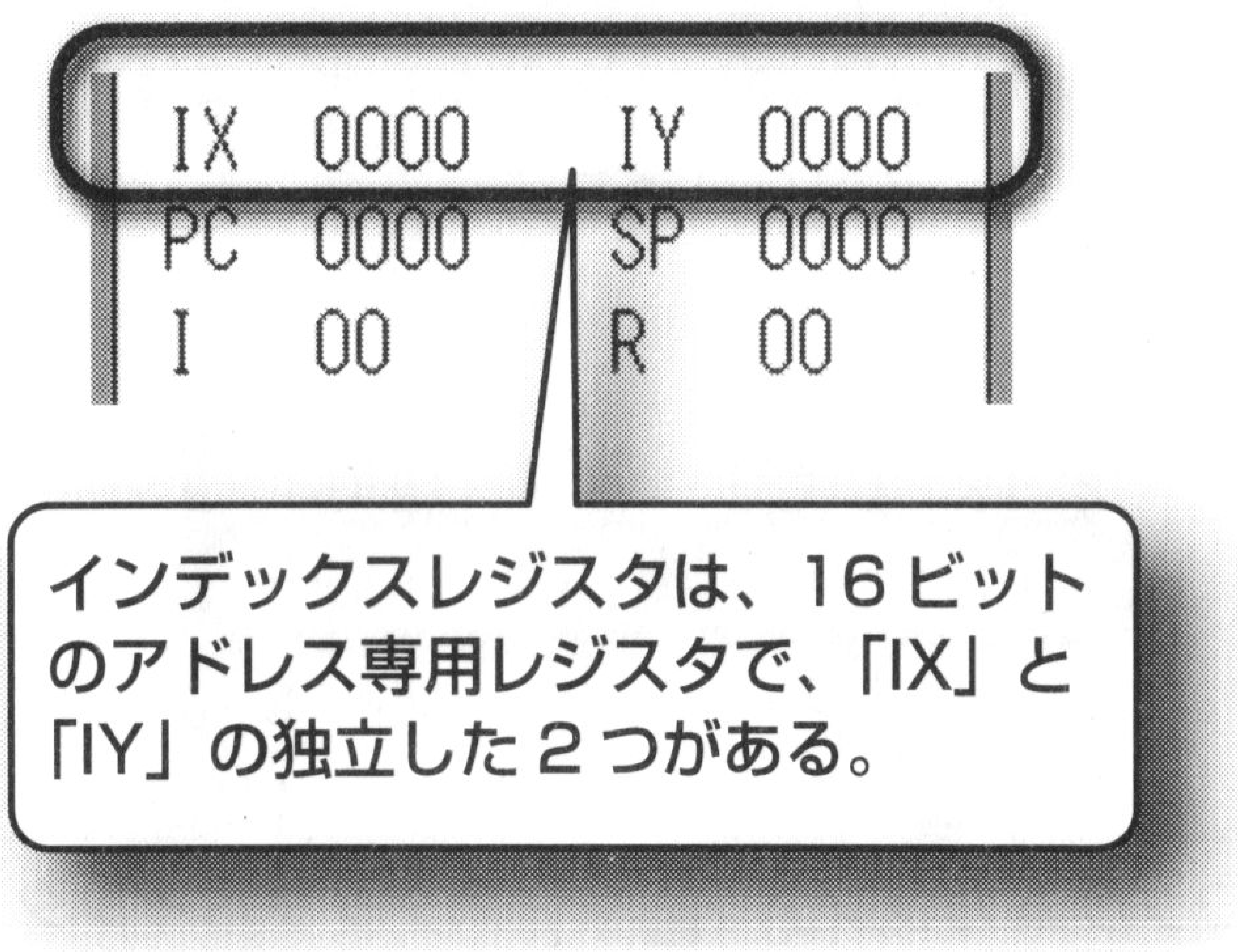


図4-12 インデックスレジスタ

■インタラプトページアドレスレジスタ（8ビット）

interruptの頭文字を取って、Iレジスタとも呼びます。インタラプトは、CPUの外からの「割り込み」のことです。

Z80の割り込み処理方法には3通りあり、その中でも代表的なのが、Iレジスタを使う「割り込みベクタ」という手法です。

たとえば、7.10節で実験するサブルーチンの動きと比べると、違いがわかるでしょう。サブルーチンでは、メインルーチンの中に必ずコール命令が含まれており、いつサブルーチンへジャンプさせるのかを、あらかじめ決めることができます。

しかし、Iレジスタを用いる割り込み処理では、メインルーチンの中にコール命令のような命令など何もありません。その代わりに、突然外部のI/Oポートから割り込みの要求があると（割り込み処理ルーチンと言う）、どこからの割り込みなのかをすみやかに特定し、CPUがどのプログラムを実行すればよいのかがわかる仕組みになっています。

なお、あとにも触れますが、付録のZ-Visionでは、この動作をわかりやすくシミュレートすることができませんので、やむを得ず割愛しました。

■メモリリフレッシュレジスタ（8ビット）

Refreshの頭文字を取って、Rレジスタとも呼びます。

かつてZ80は、パソコンのCPUとして使われました。Rレジスタは、DRAM（ダイナミックRAM）のメモリリフレッシュを自動的に行うための専用レジスタです。

RAMには、大きく分けてSRAM（スタティックRAM）とDRAMがあり、コンデンサに電気を蓄える原理でデータを記憶しているのがDRAMです。

ところが、このコンデンサ内の電気エネルギーはひじょうに微少で、放置しておくと無くなってしまい、データが壊れてしまいます。そこで、一定の周期でDRAMに同一のデータを重ね書きする必要があるのです。これをメモリリフレッシュと言います。

通常プログラマは、Rレジスタを対象にプログラミングしませんので、本書でも触れていません。

ところで、Z-Visionでプログラムを走らせると、Rの値が勝手に変化することがあります。これが、自動的に動作している様子です。実は、かつてZ80がパソコンのCPUで使われたころ、Rレジスタの値を乱数として利用したことがありました。

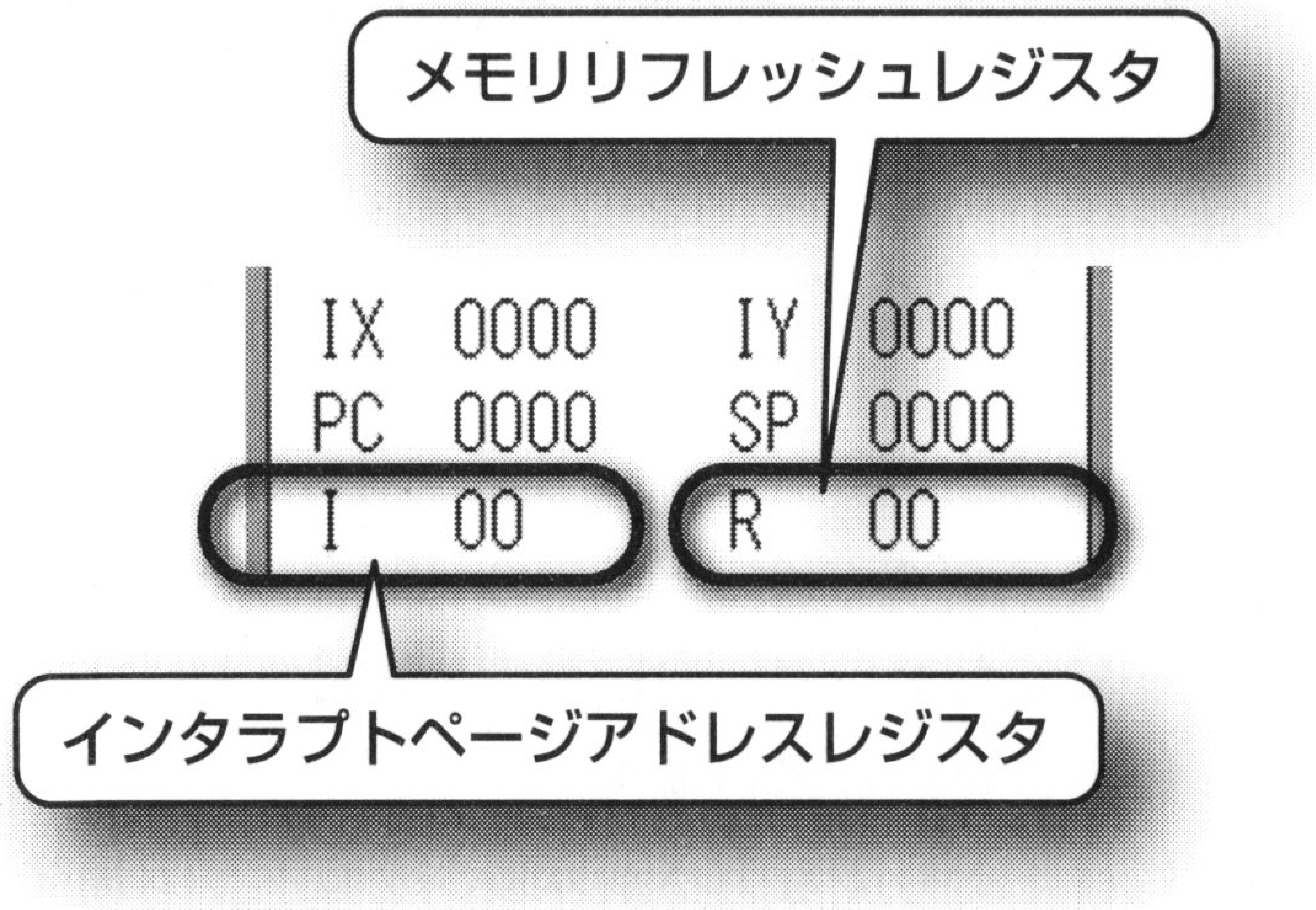


図4-13　IレジスタとRレジスタ

第5章　プログラムを走らせる

第４章で、実際のCPUとZ-Visionの画面との対応を見ていただきましたが、表示内容が少し物足りないと感じた読者がおいでになると思います。

その理由は、そもそもデバッガは、プログラム開発を効率的に進めるためのツールですので、CPU内部のありとあらゆる場所の値を知る必要がないからです。

つくったプログラムをシミュレートすることによって、できるだけ早くバグを取り除くことが最大の目的ですので、前章の表示内容で必要十分と言えます。

それを裏付けるため、これからテストプログラムを実際に走らせてみることにします。Z-Visionの画面上で高速実行することもできますが、これは参考程度に見ていただきます。

むしろ、初めて命令語の機能を理解するときには、命令文を１行ずつ実行し、その都度、レジスタの内容がどのように変わるのかを観察するのがよいでしょう。

5.1　高速実行してみる

現実のコンピュータの動作速度は、人間の世界とはかけ離れています。まず、その様子を画面で見てみましょう。なお「高速実行」は実際のCPUの速度で動くわけではなく、レジスタウィンドウの変化を確認するための機能です。

①「高速実行」ボタンをクリックします。

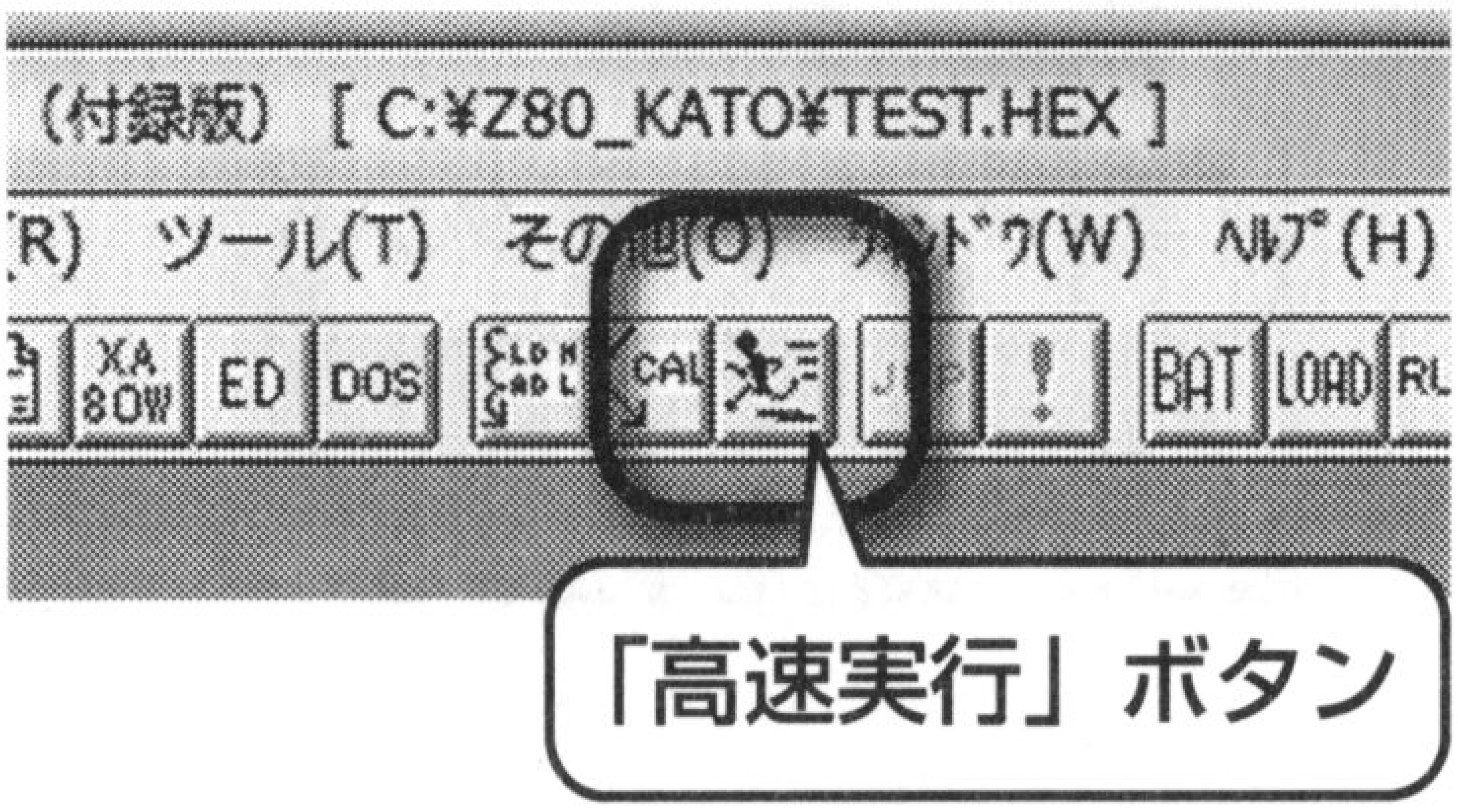


図5-1　高速実行する

②逆アセンブルウィンドウは静止したままですが、レジスタウィンドウは忙しく数字を変化させます。

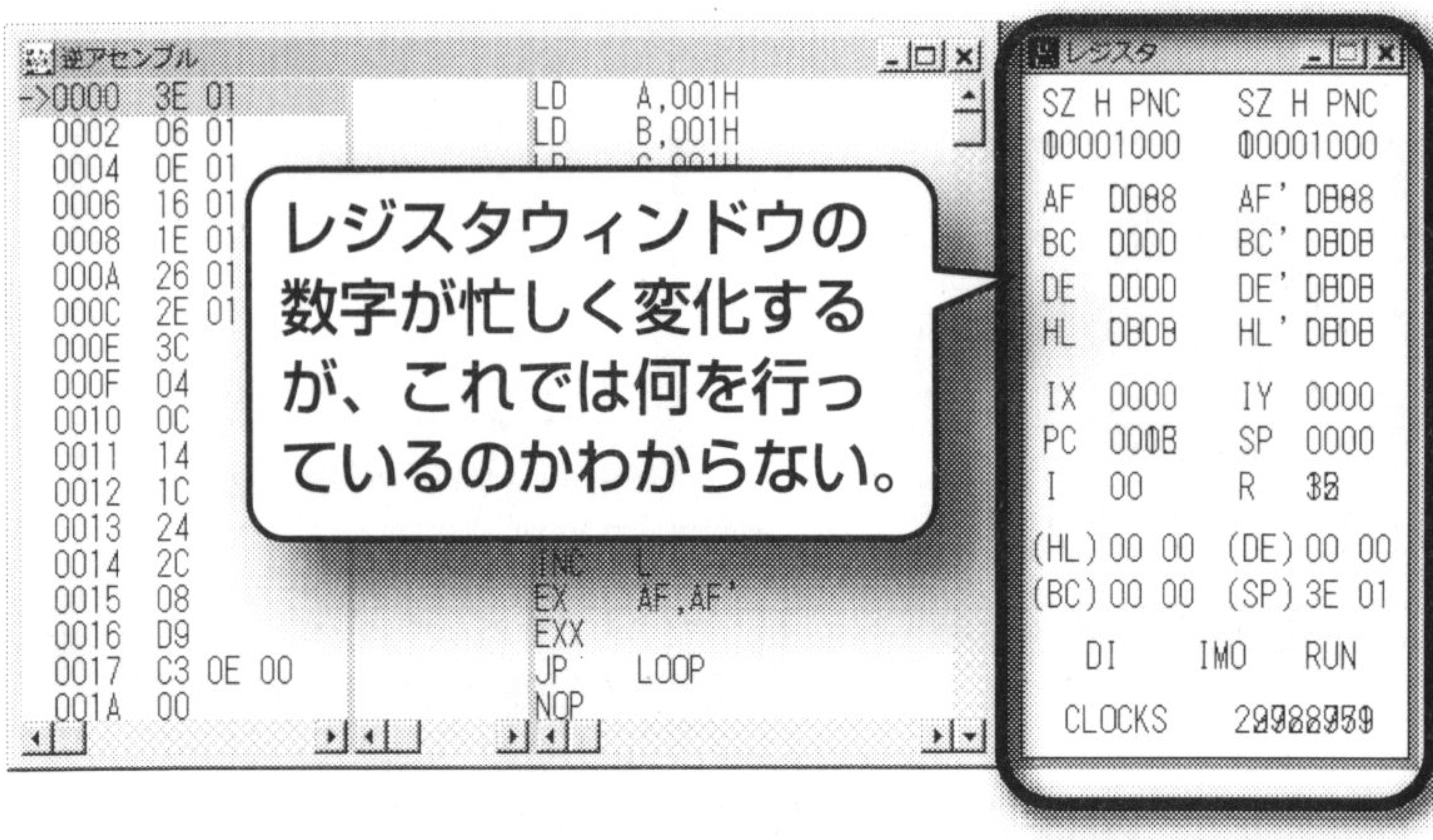


図5-2　高速実行中

③しかし、これではCPUが何をしているのかが、まったくわかりません。いったん実行を止めましょう。
「！」ボタンをクリックするか、実行メニューから「実行の停止」を選びます。

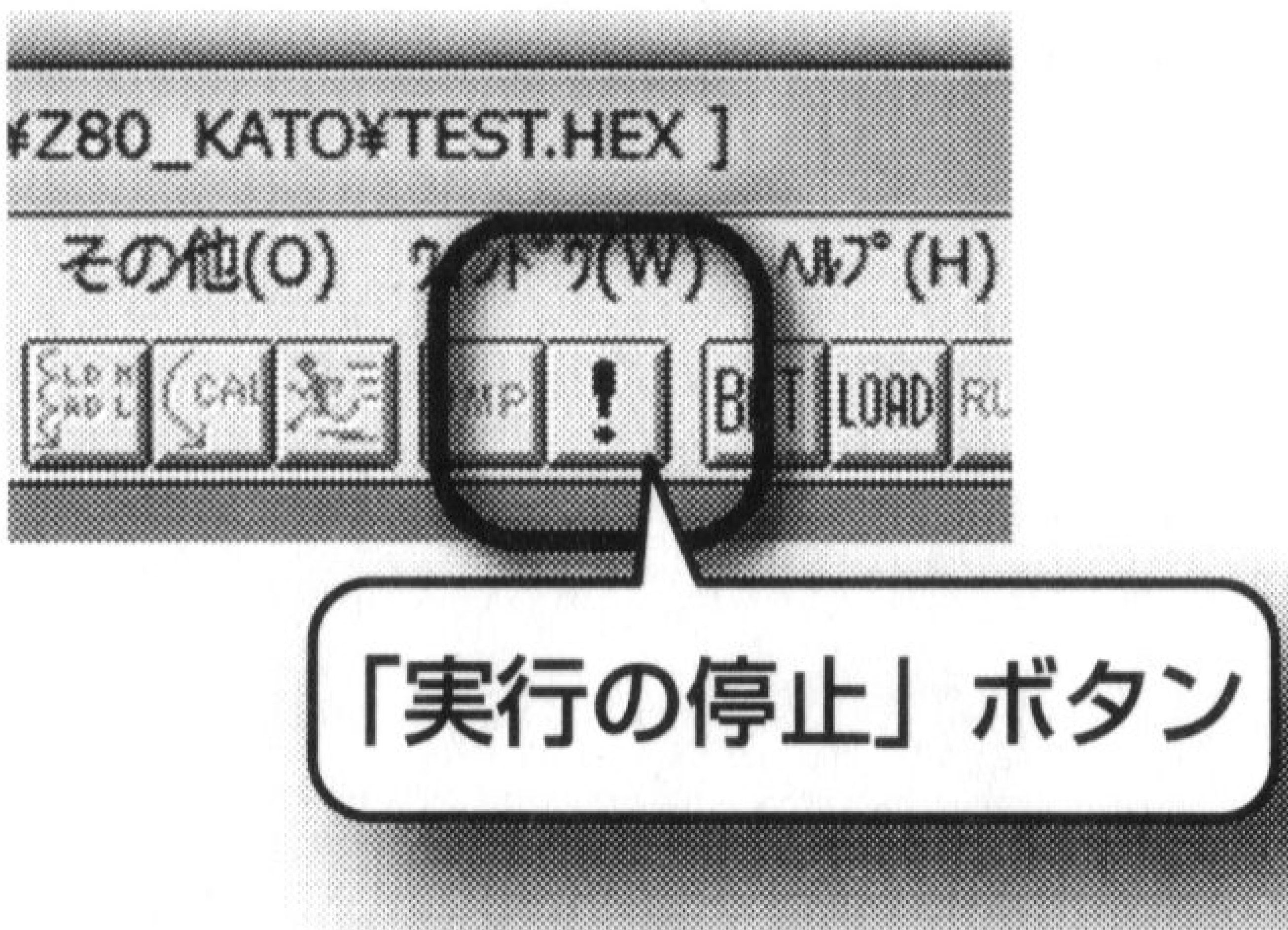


図5-3　実行を停止する

④レジスタウィンドウの中は、意味不明な数字で埋まっていますので、リセットしましょう。
「RESET」ボタンをクリックすると、「レジスタのリセット」ダイアログボックスが現れますので「OK」ボタンをクリックします。



図5-4　レジスタをリセットする

⑤すると、レジスタウィンドウの中の数値が、すべてゼロにリセットされ、逆アセンブルウィンドウの緑色の帯は、「->0000　3E　01」に戻ります。

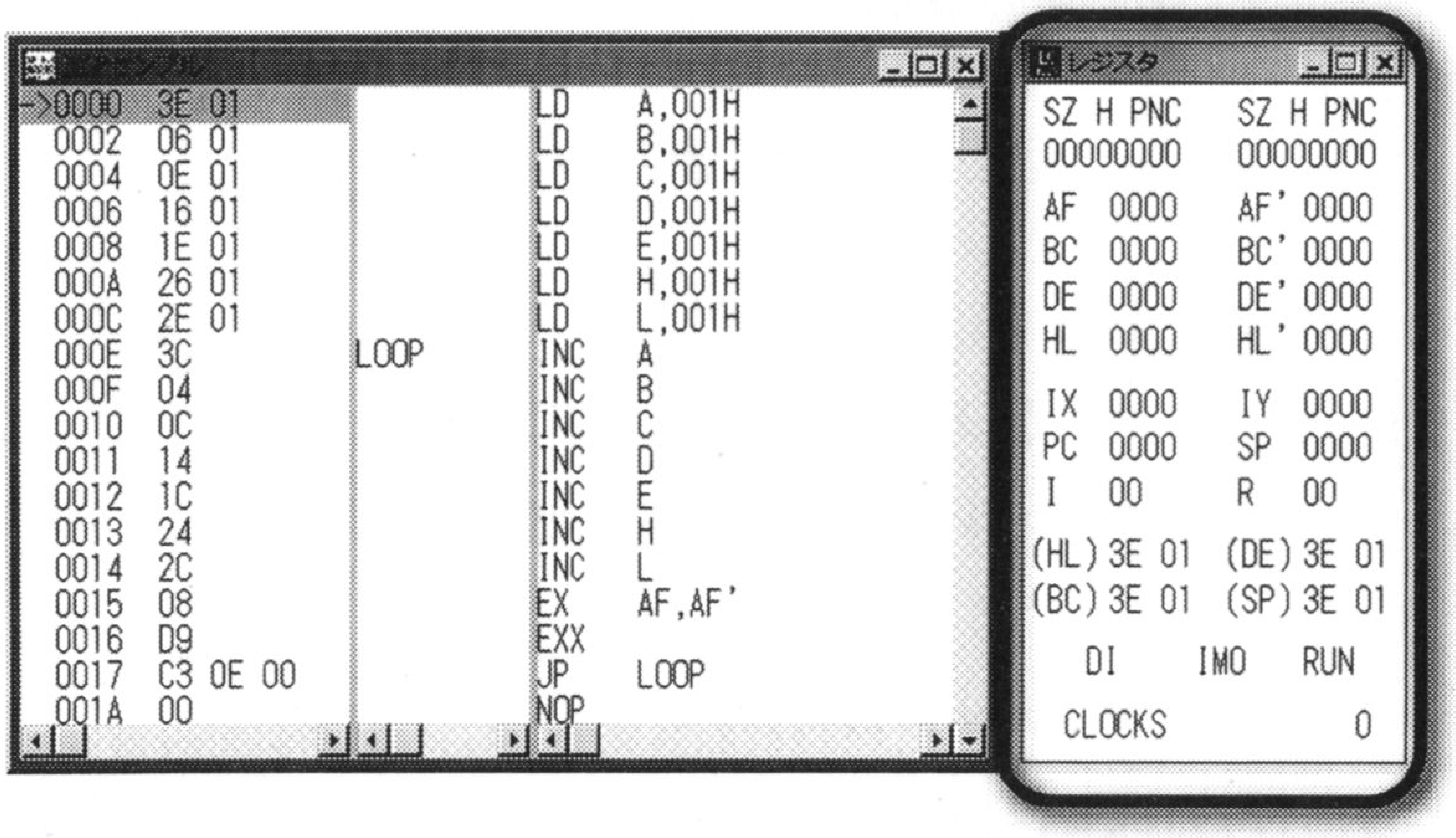


図5-5　レジスタがすべてリセットされる

5.2　1行ずつ実行して観察する

今度は、プログラムを1行ずつ実行させます。それぞれの命令語の動きが、レジスタウィンドウによって把握できます。

①「1行トレース」ボタンをクリックすると、プログラムを1行だけ実行します。

しかし、単にクリックを繰り返すのでは意味がありません。その都度、逆アセンブルウィンドウとレジスタウィンドウをじっくりと観察してください。また、変化を見落としても「RESET」ボタンをクリックすれば、いつでも初期状態に戻せます。

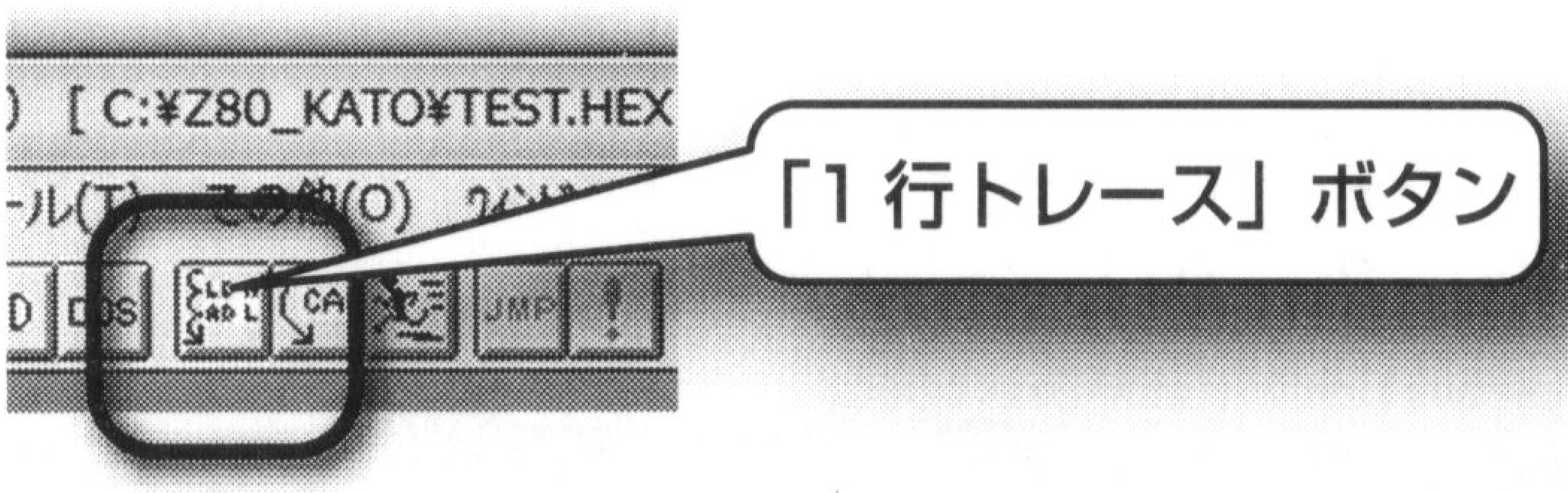


図5-6　1行だけ実行する

②1行目の「LD　A,1」の実行によって、アドレスは+2され、0002Hに進みました。これは、ロード命令「LD A,1」を実行するために2バイト必要としたことを意味します。

一方、レジスタウィンドウでは、主レジスタセットのAFの値が0100Hに変化しています。Aレジスタに01Hが入力されたということです。

もう1つの変化は、「CLOCKS」の値です。最初0だった値が7に進みました。これは、CPUに供給されるクロックが7周期必要だったことを意味します。



図5-7　1行だけでも変化は大きい

③同様にして、どこが変化していくのかを順次追ってみましょう。

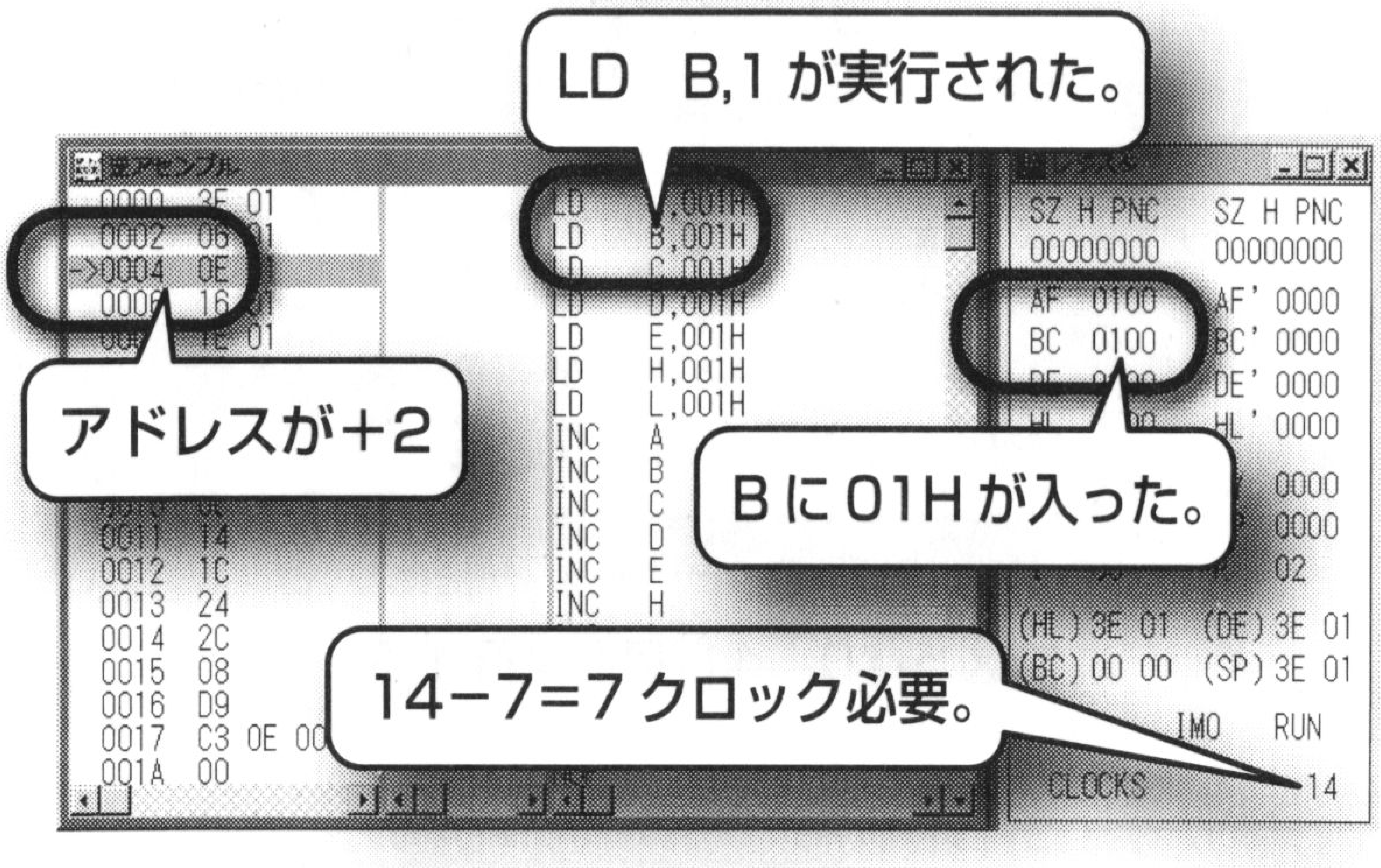


図5-8　2行目を実行する

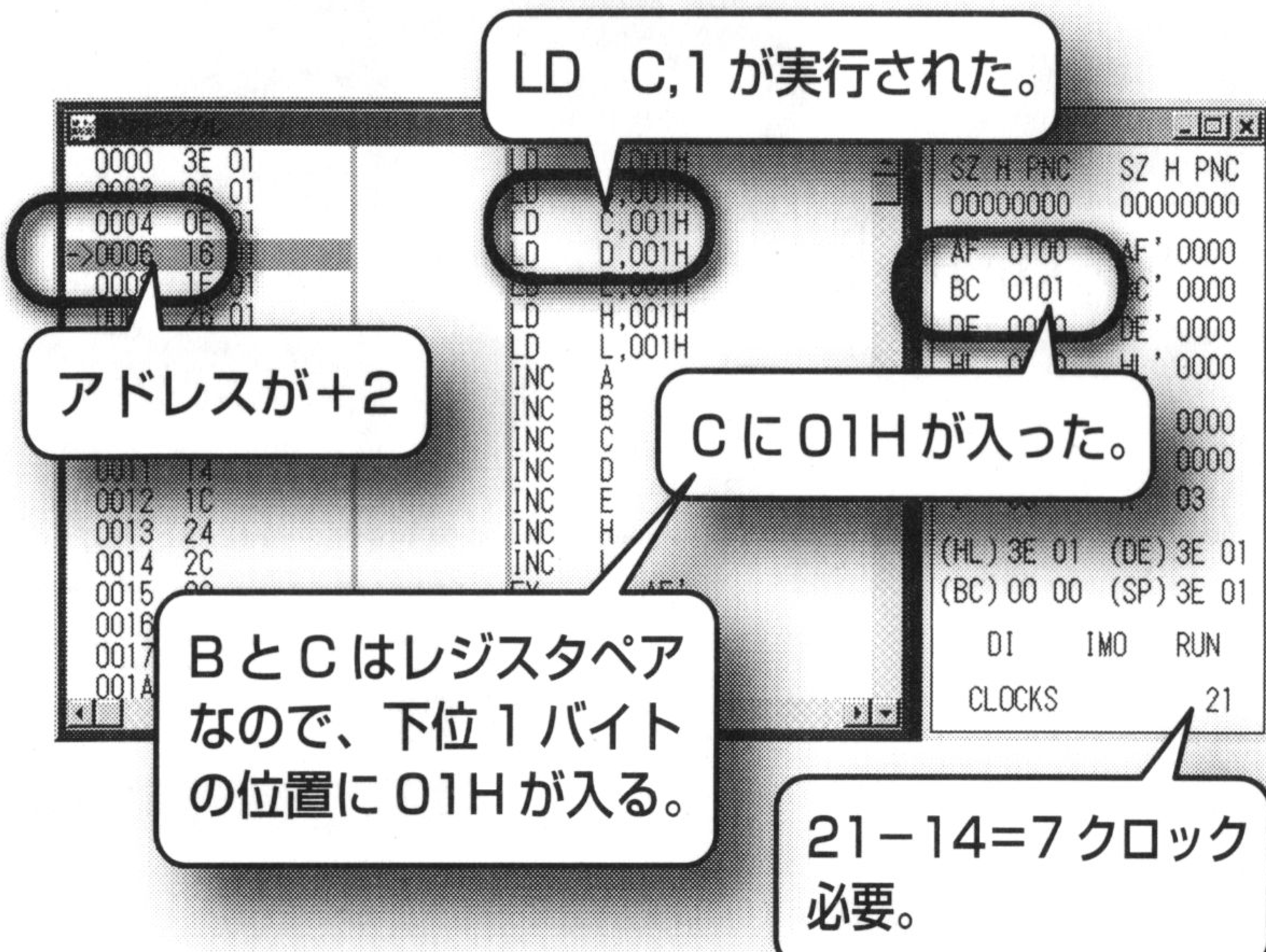


図5-9　3行目を実行する

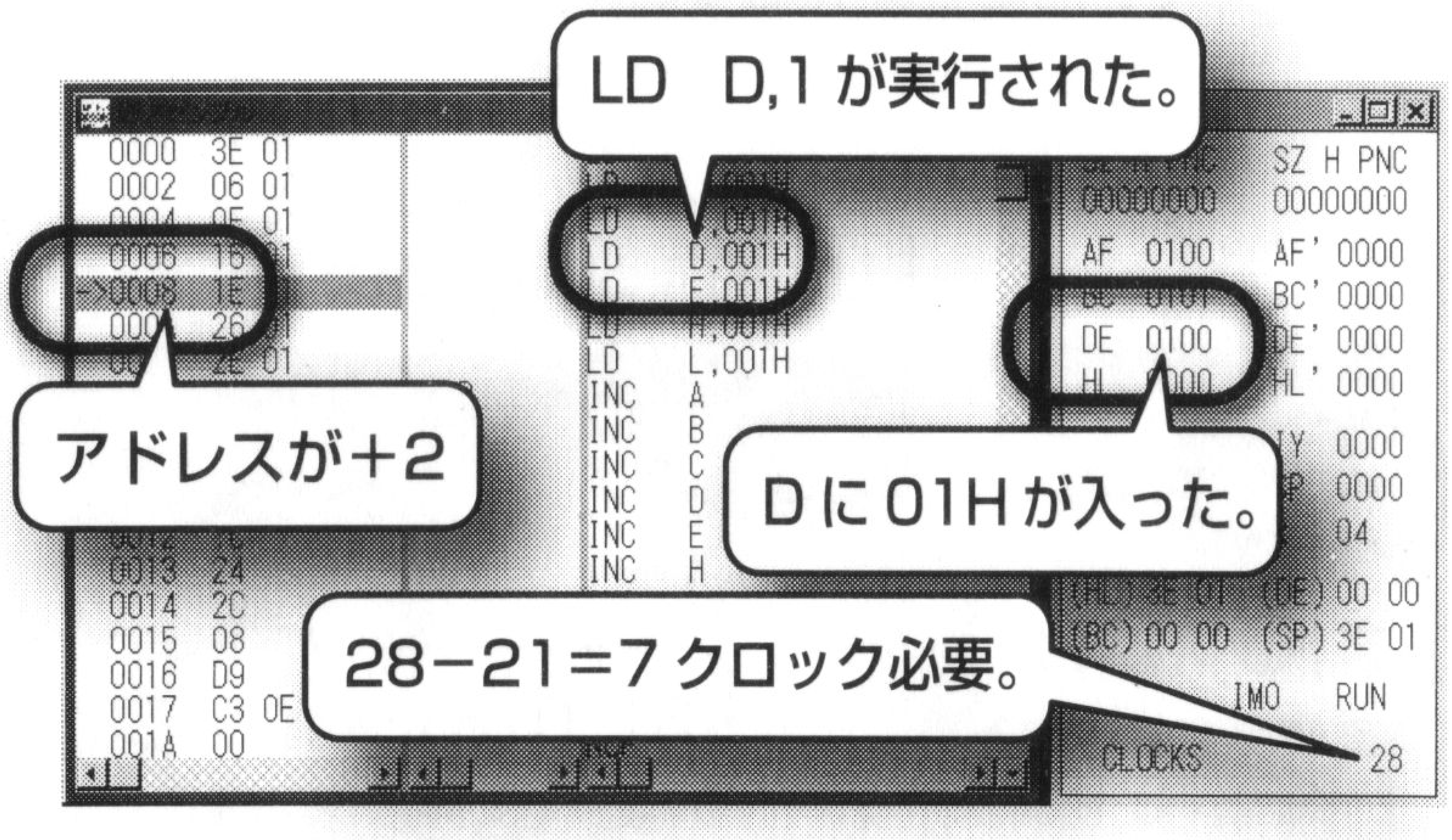


図5-10　4行目を実行する

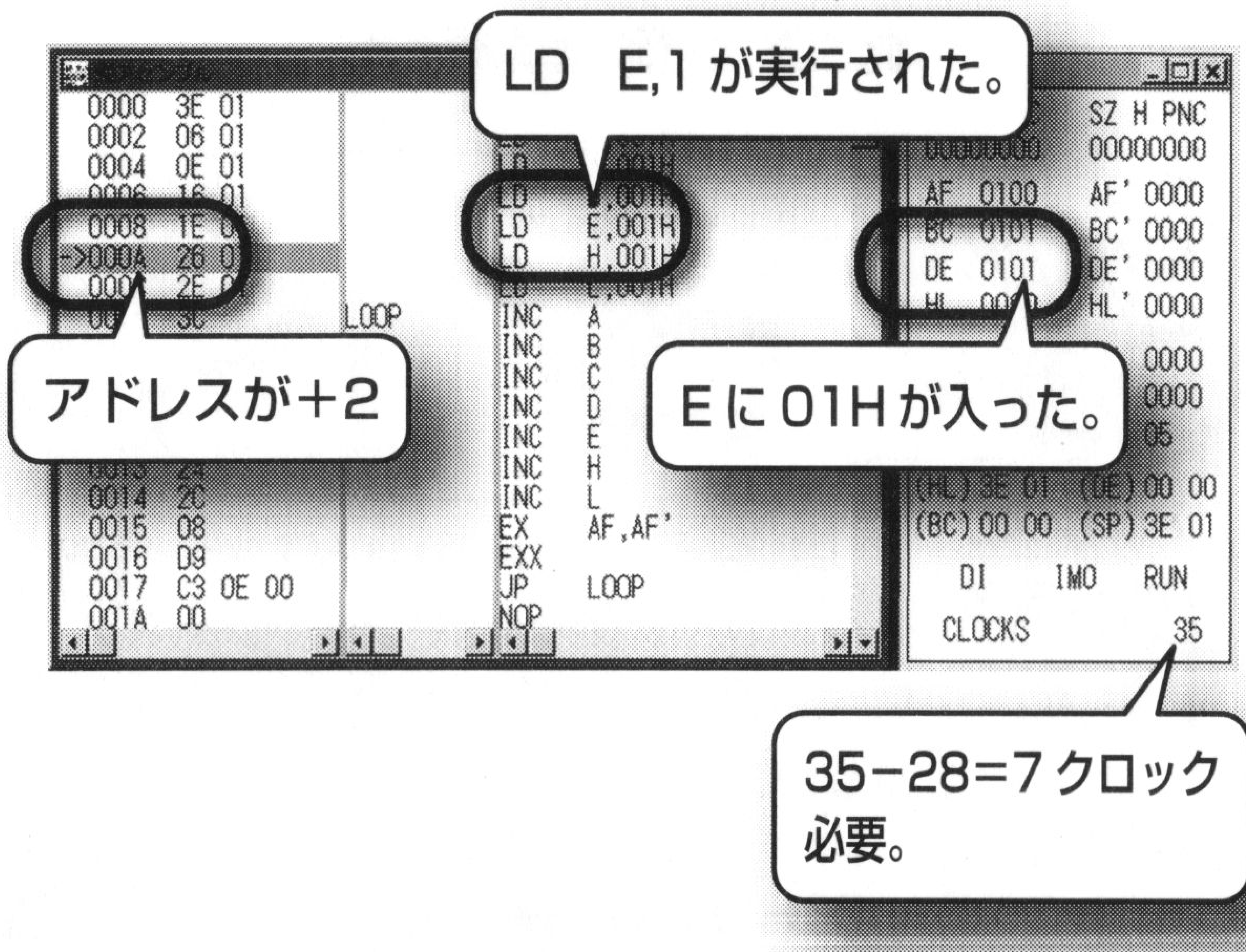


図5-11　5行目を実行する

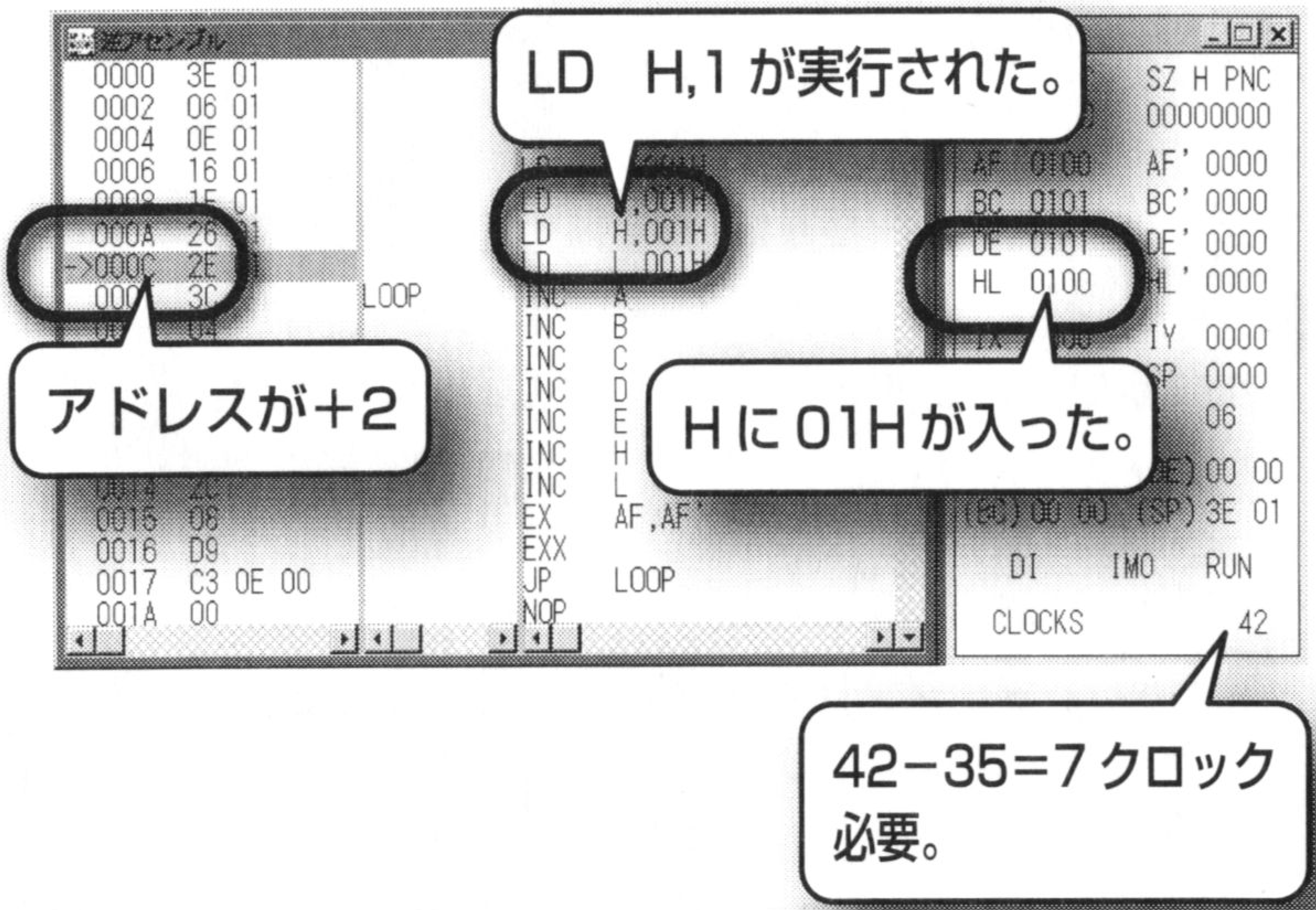


図5-12　6行目を実行する

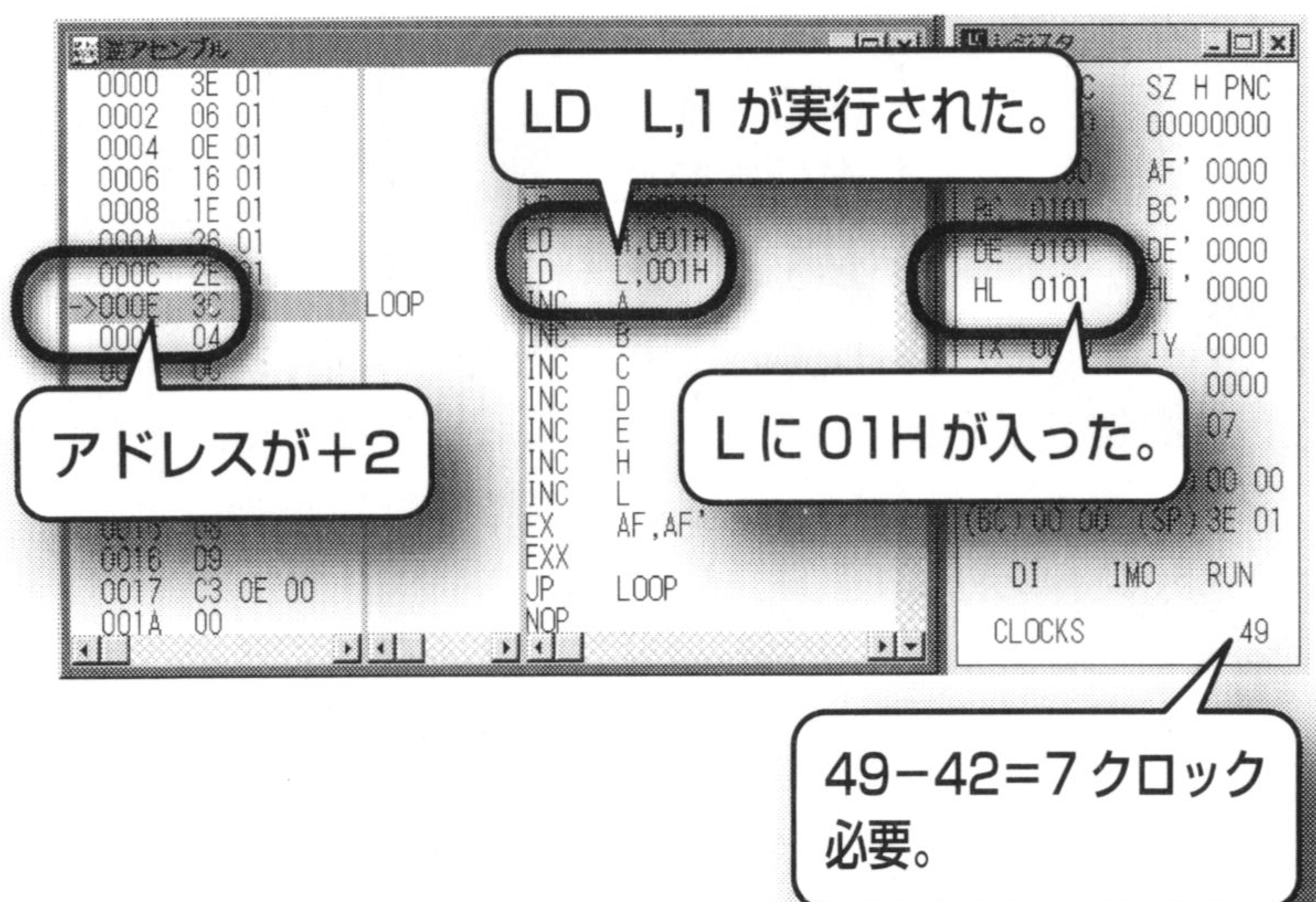


図5-13　7行目を実行する

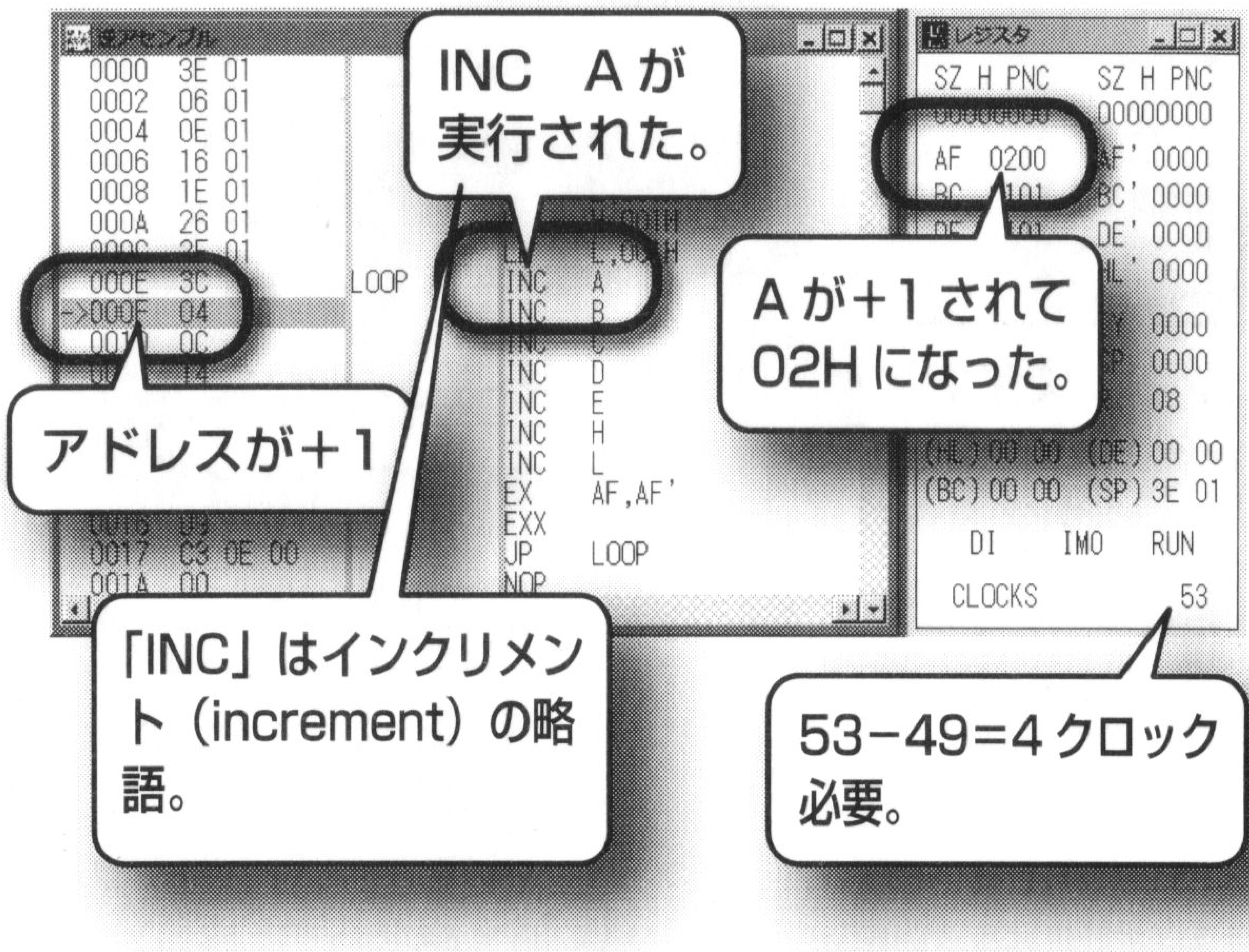


図5-14　8行目を実行する

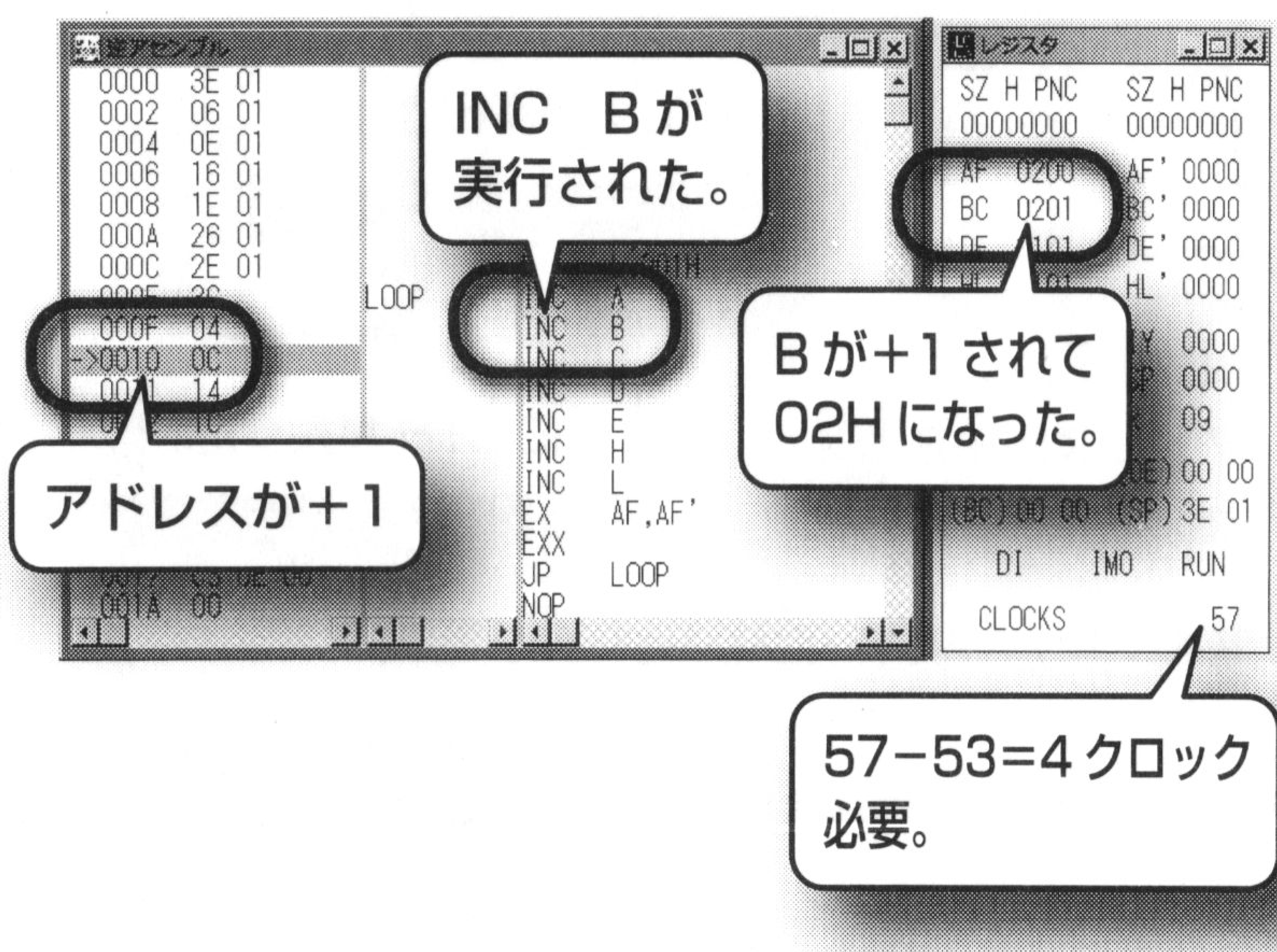


図5-15　9行目を実行する

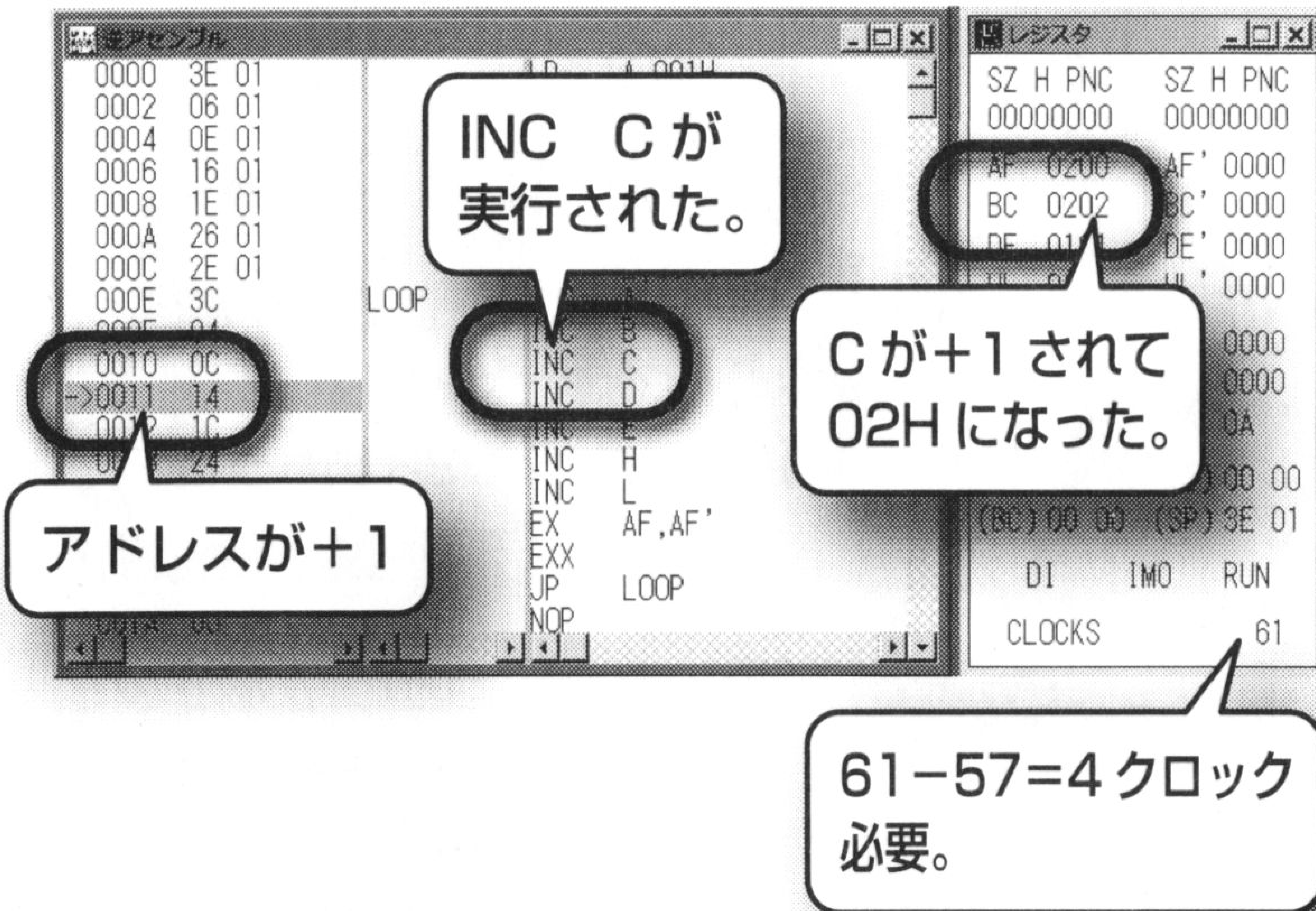


図5-16　10行目を実行する

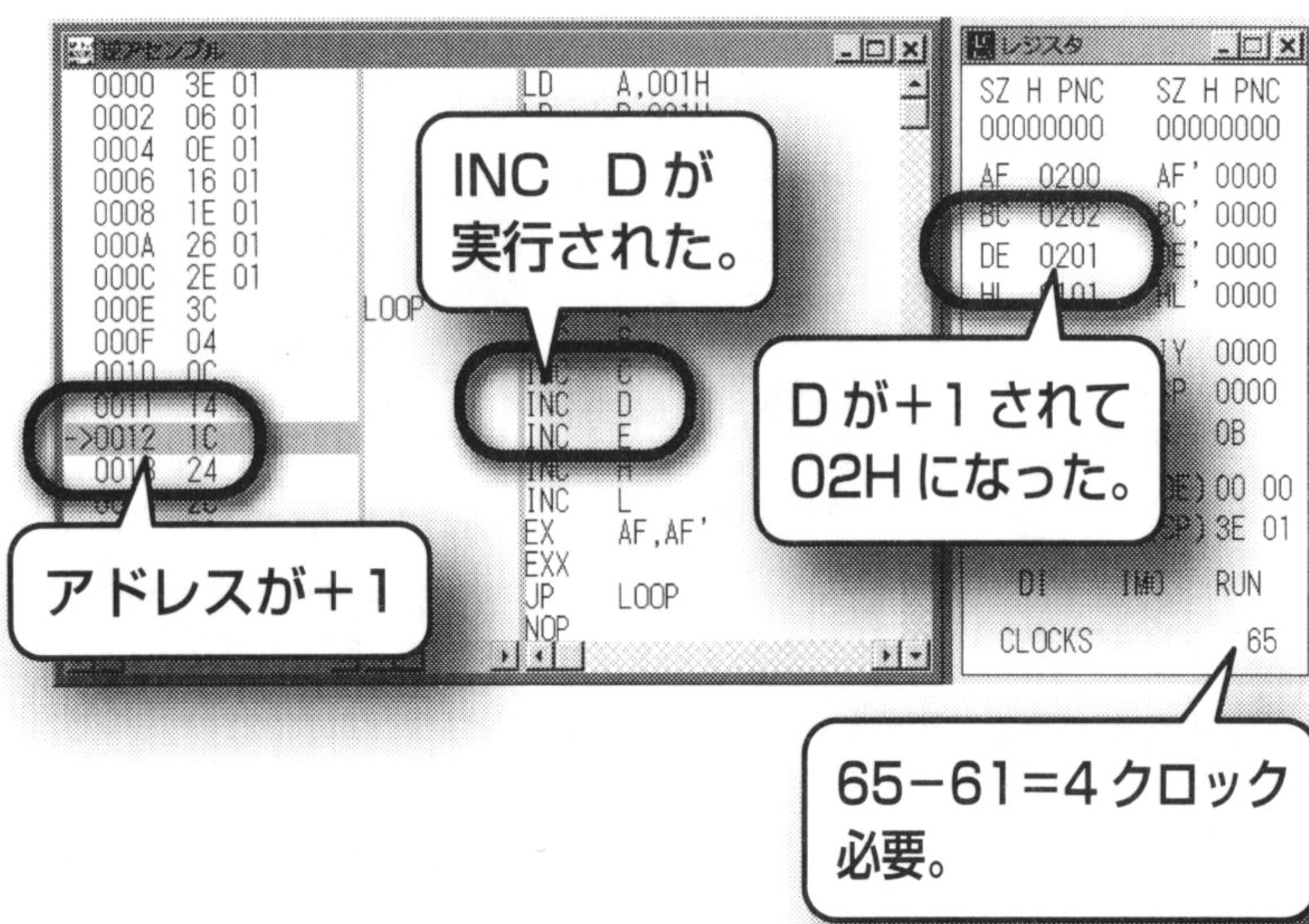


図5-17　11行目を実行する

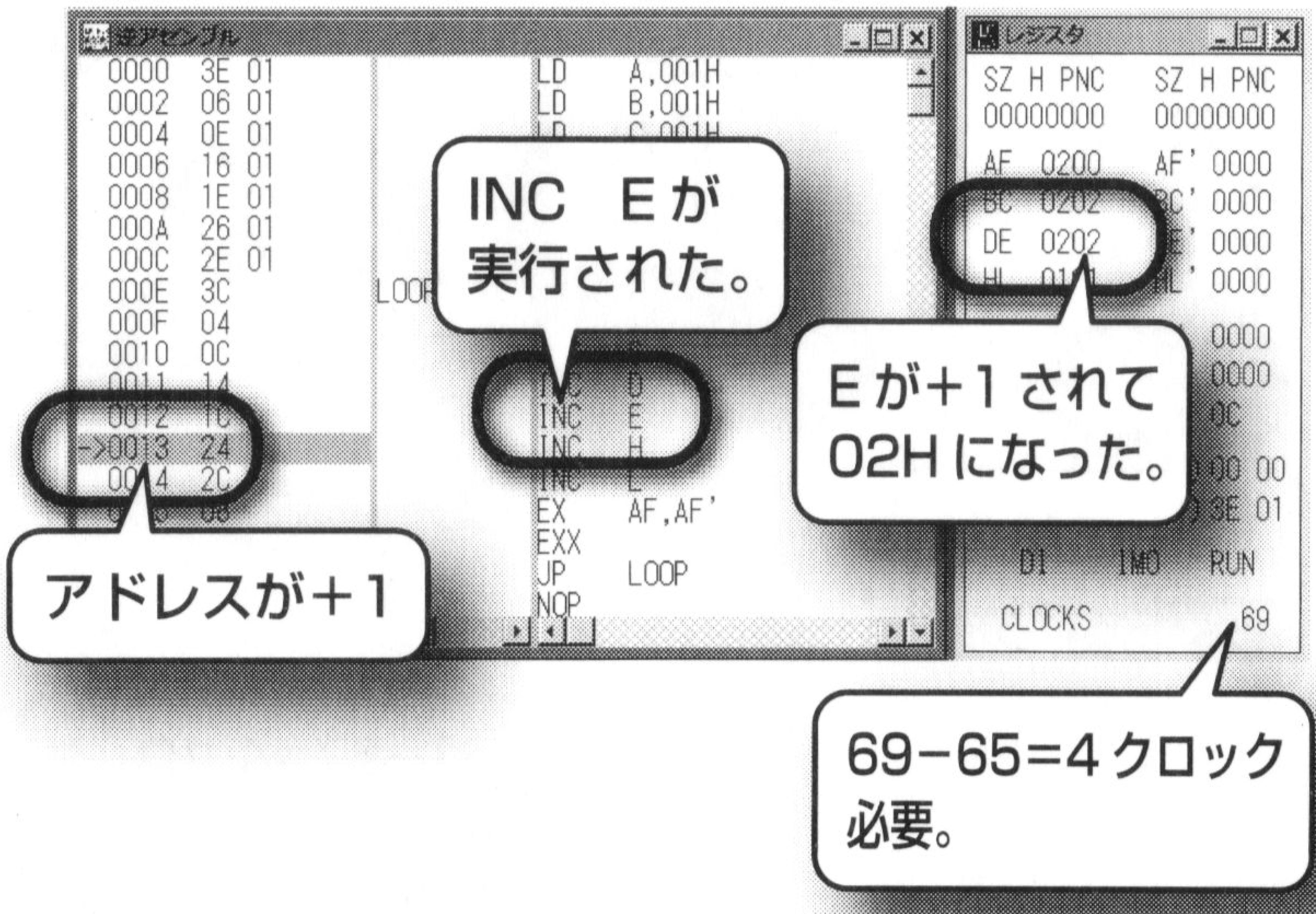


図5-18　12行目を実行する

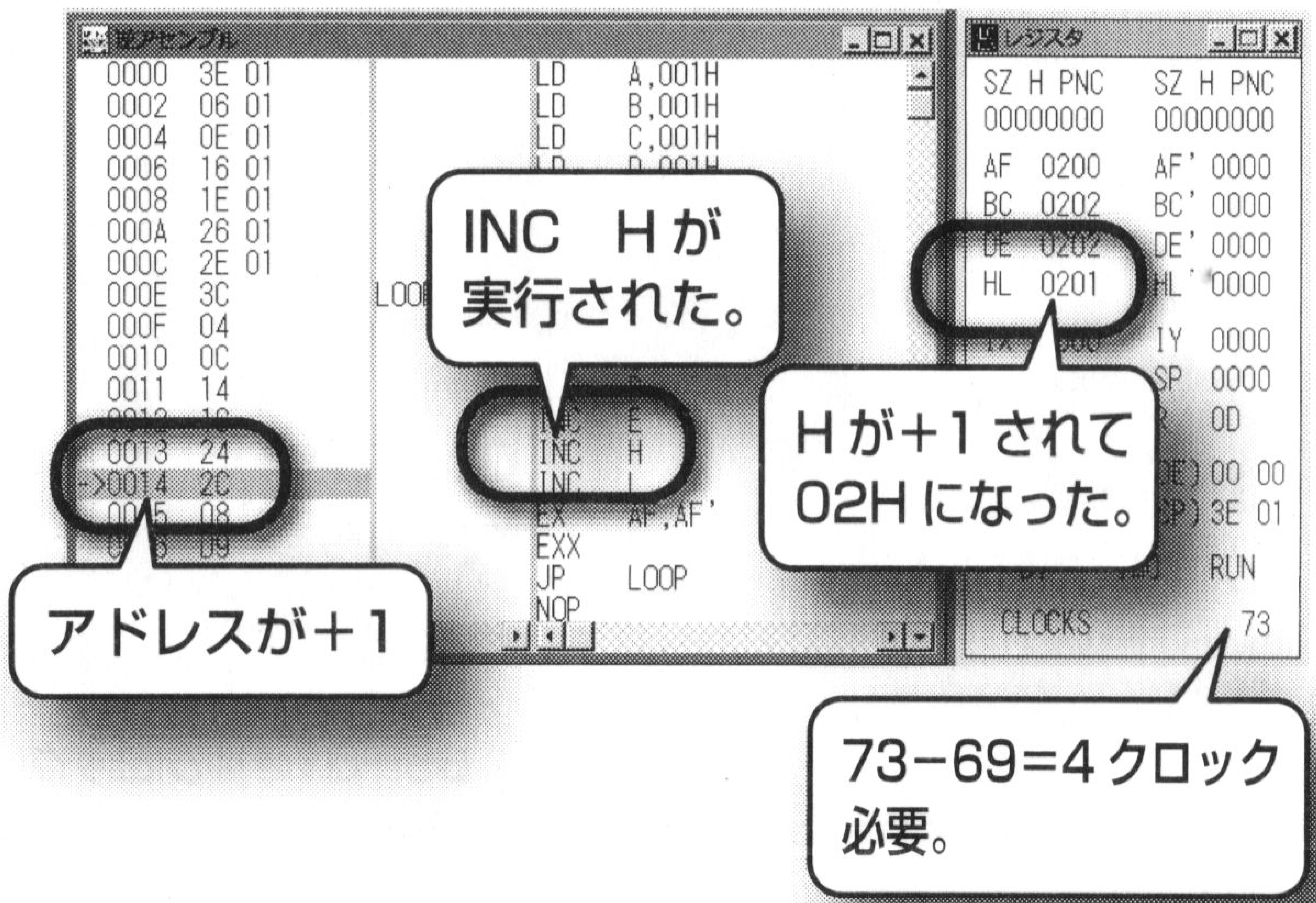


図5-19　13行目を実行する

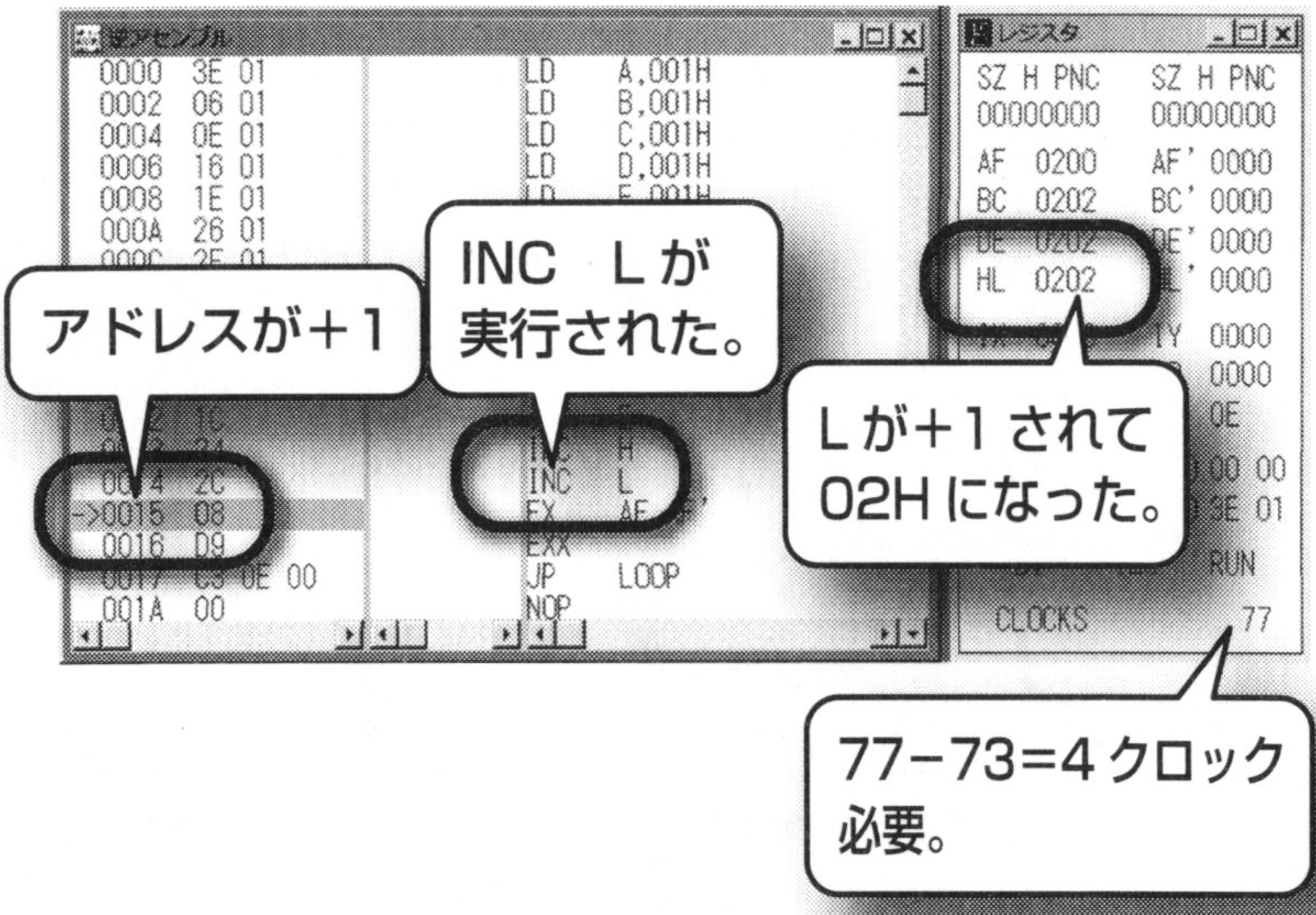


図5-20　14行目を実行する

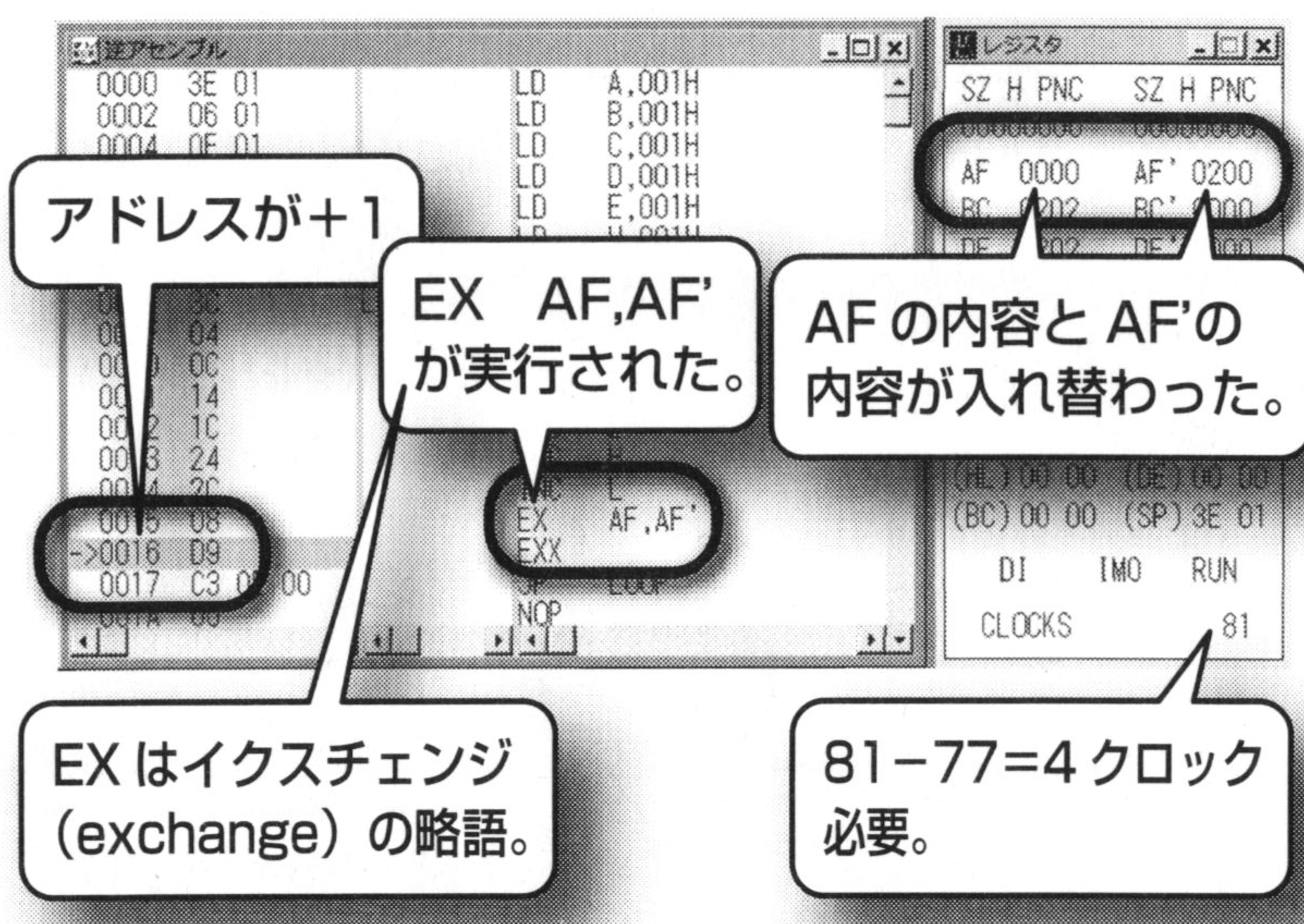


図5-21　15行目を実行する

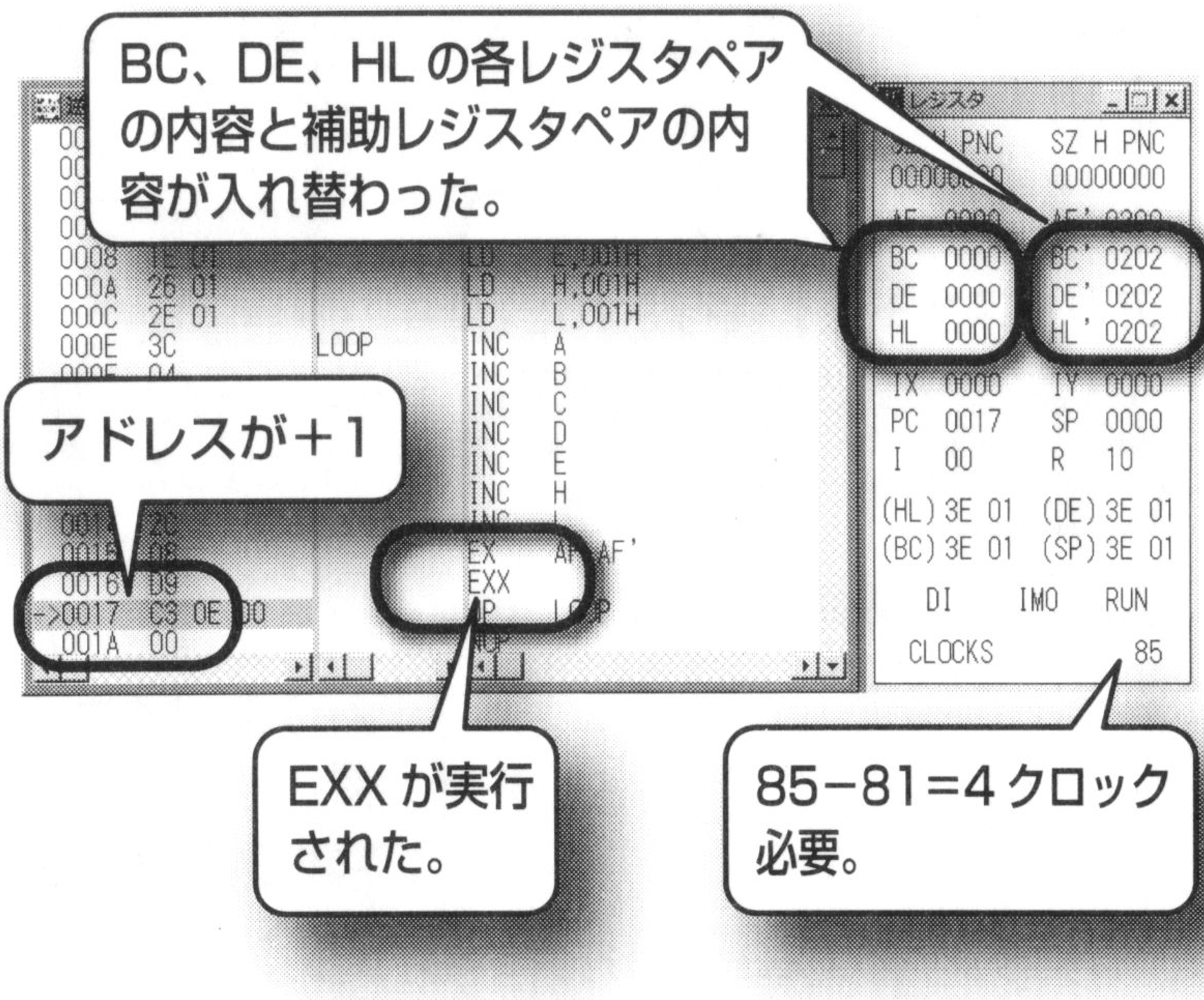


図5-22　16行目を実行する

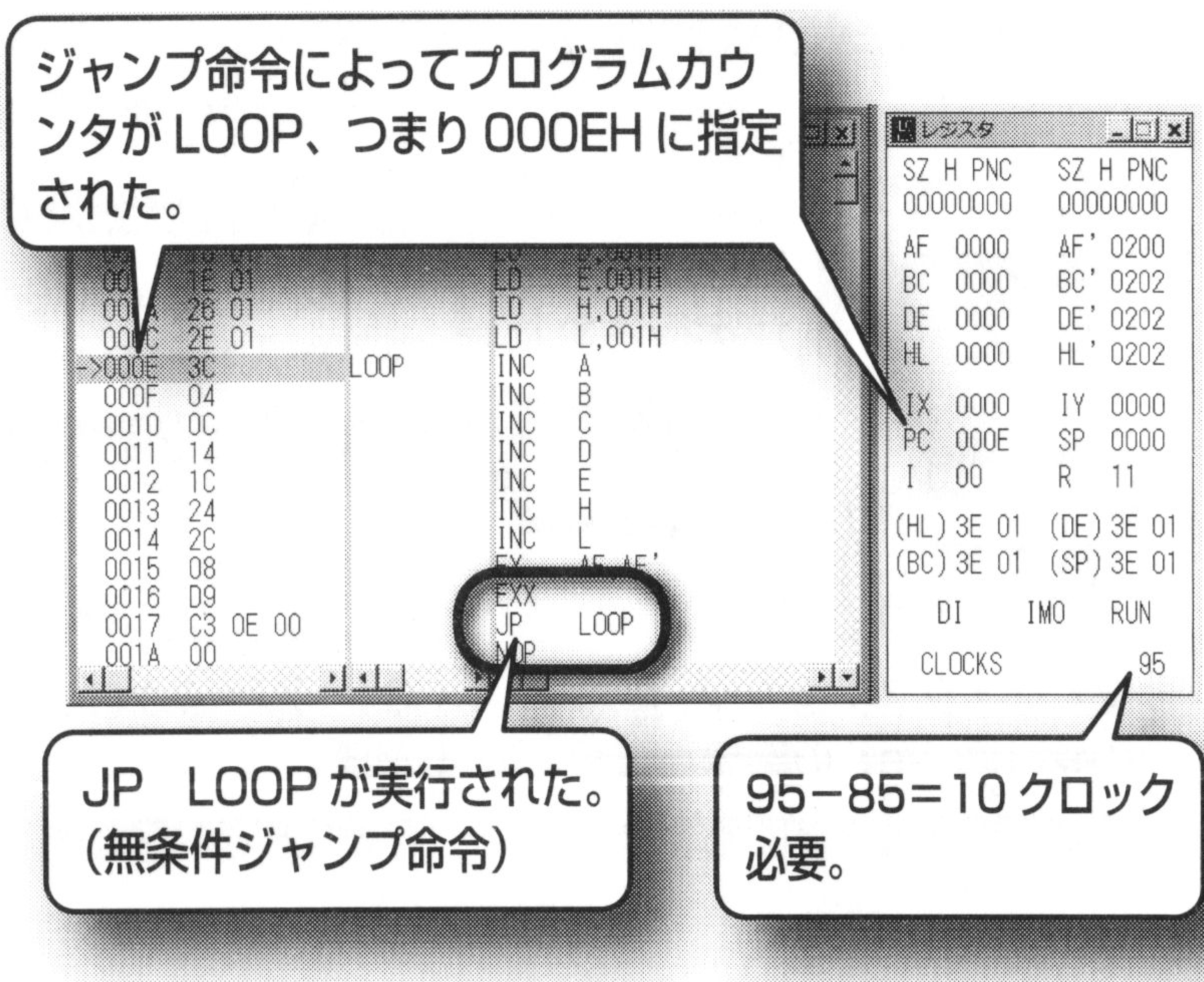


図5-23　17行目を実行してジャンプする

④このプログラムは永久ループですので、強制的に停止しないかぎり、８行目から17行目までの間の実行を永久に繰り返します。

ここまでシミュレートすれば、あとの動作は容易に推測できるでしょう。また、Z-Visionを用いたシミュレートのしかたをご理解いただけたかと思います。

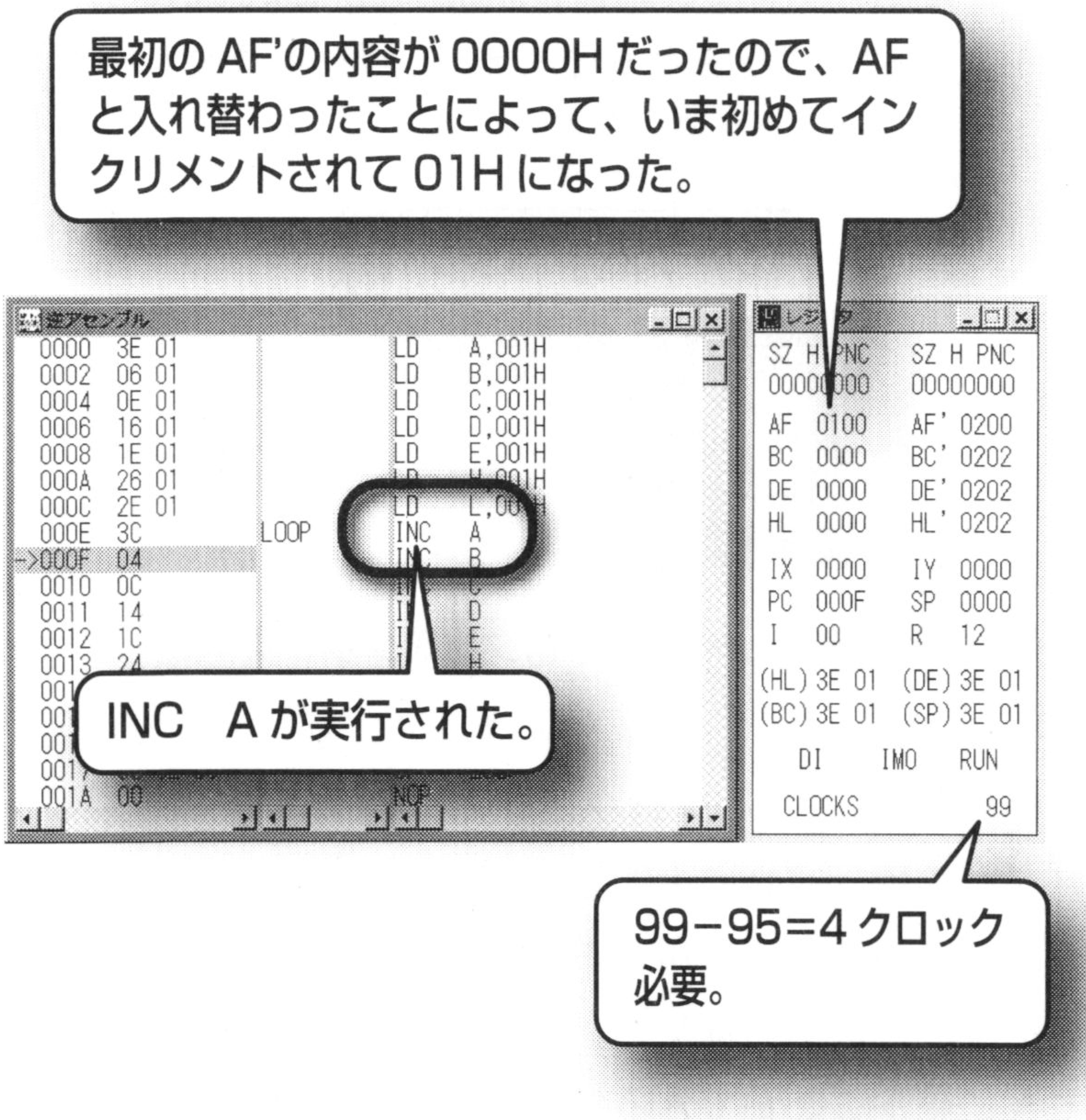


図5-24　8行目を実行する（2回目）

5.3　ソース行ウィンドウの併用

逆アセンブルウィンドウは、実際にZ80が実行する機械語だけを抽出して、元のプログラムリスト（ソース・ファイル）を復元して表示しています。

そのため、3.1節で入力したとおりの内容にはなっていません。しかし、つくったプログラムが正常に動作するかどうかを確認するのであれば、余計な情報は不要です。

どうしても気になるようでしたら、「ソース行」ボタンをクリックしてみてください。

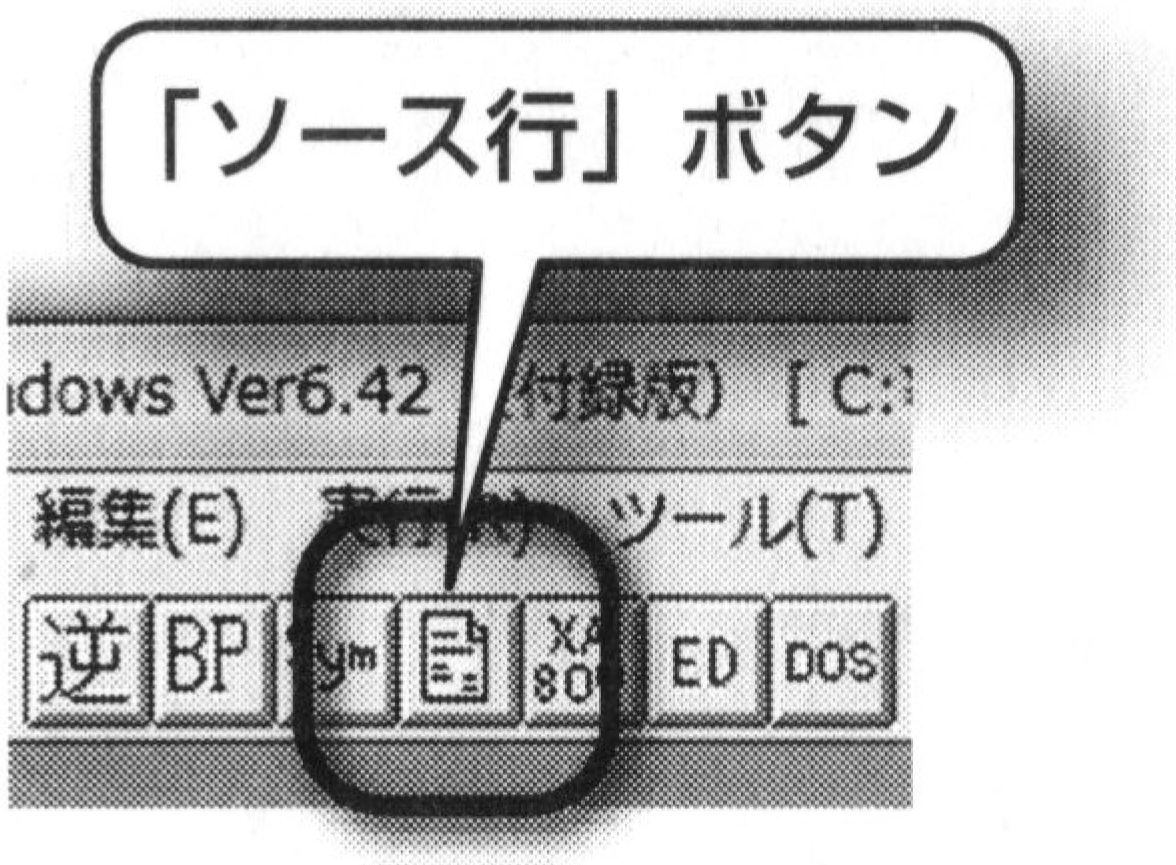


図5-25　「ソース行」ボタンをクリックする

図5-26に示すソース行ウィンドウが開きます。このウィンドウでは、初めに入力したプログラムリスト、つまりソース・ファイルの記述どおりに表示してくれます。

しかも、行数やアドレス、機械語などあらゆる情報をいちどきに見ることができるだけではなく、逆アセンブルウィンドウと同様に、1行ずつ実行している様子も表示してくれます。

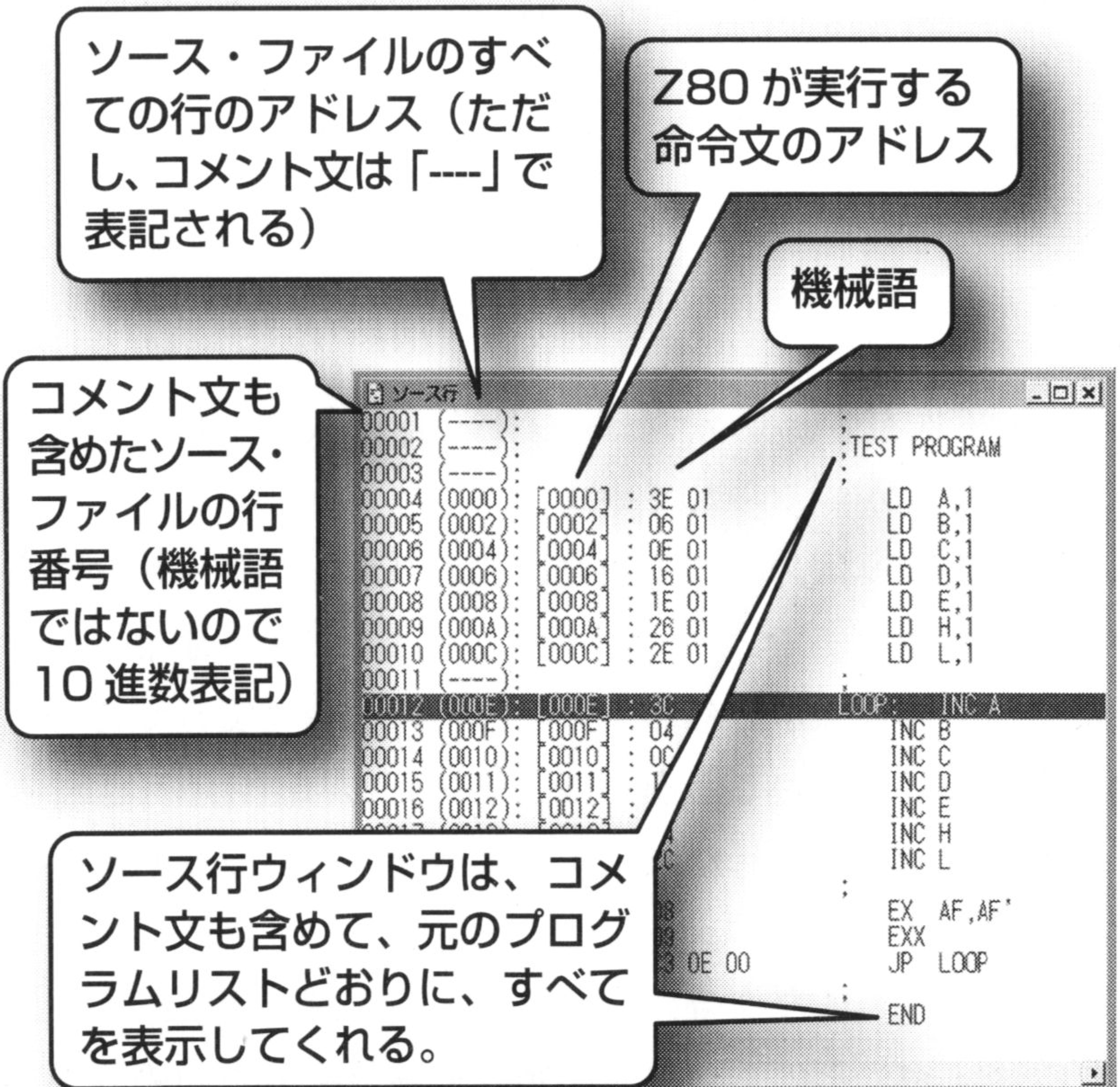


図5-26　ソース行ウィンドウの表示

第6章　CPUのことばを探る

Z80では、8ビットつまり1バイトの数値を基本単位にして、命令がつくられています。

その仕組みは巧みに分類されており、1バイト命令から4バイト命令まであります。たとえば4バイト命令とは、4バイトの数値がそろわないと、その命令を実行することができない命令のことを意味します。

また、命令の実行結果によって変化するフラグが重要な役割を担っています。

個々の代表的な命令については、第7章で実験していただきますが、その1つ1つの動作に納得するための予備知識を、この章で学びます。

6.1　Z80命令の種類

Z80の命令数は、基本命令だけで158種あります。レジスタの違いによるバリエーションまでを含めて考えると、数が膨れ上がり、696種となります。

また、これらの命令は、図6-1のように決まった一定の形式で構成されています。

以下、専門用語が並びますが、シミュレータを動かす際には必要ありません。参考程度にお読みください。

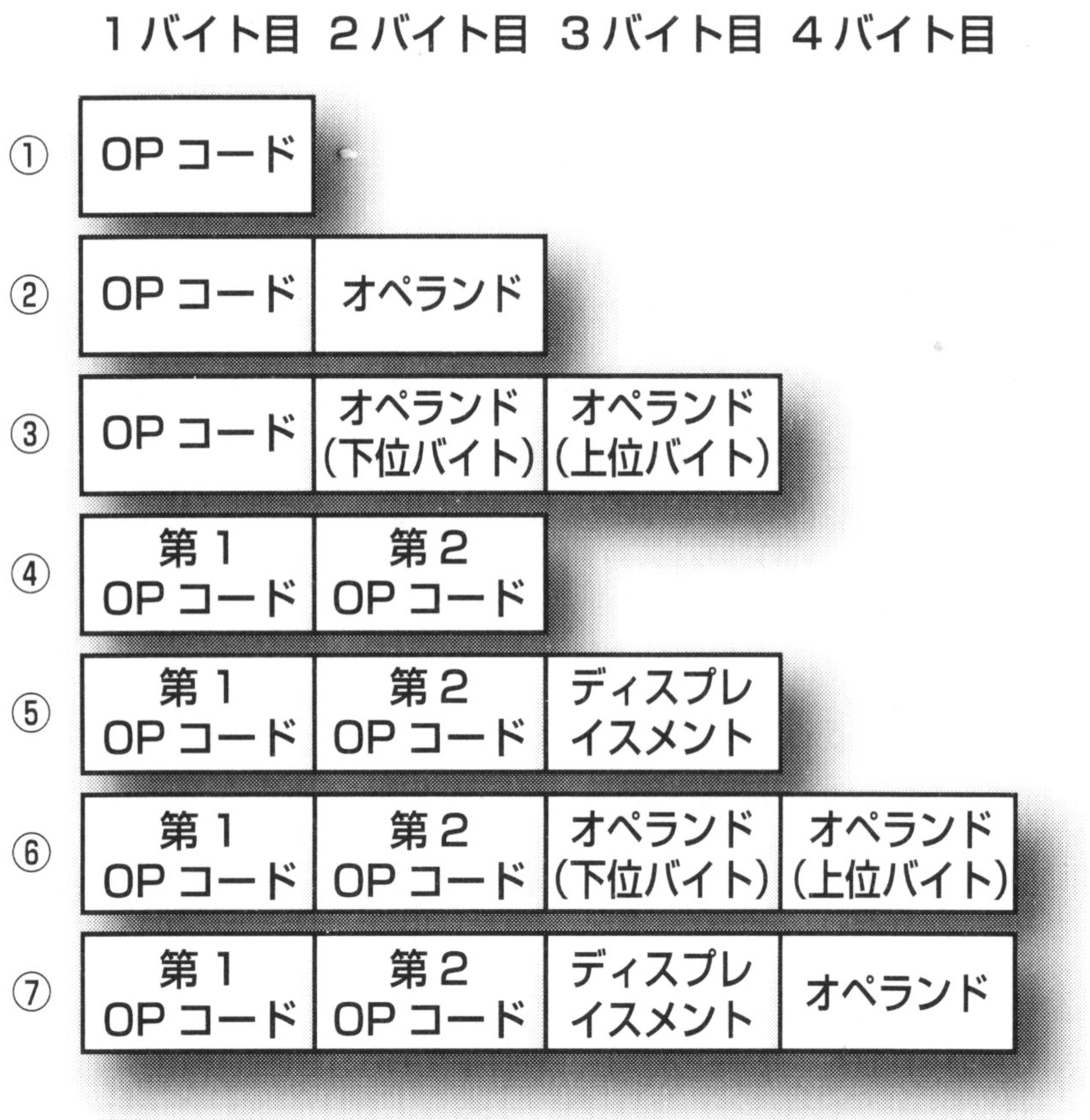


図6-1　Z80のさまざまな命令構成

①1バイト命令

OPコード（オペコード）1つだけのシンプルな命令です。

②2バイト命令（その1）

OPコードと1バイトのオペランドからなります。オペランドは、イミディエイトデータや、ジャンプ先に対する相対アドレス（現在のアドレスと目的地のアドレスの差）のことをさします。

ここで、イミディエイトデータ（immediate data）とは「即値」つまり直接の数値のことです。

③3バイト命令（その1）

OPコードと2バイトのオペランドからなります。2バイトのオペランドがある場合には、オペランドの下位バイトが先で、上位バイトが後になります。

2バイトからなるオペランドは、イミディエイトデータや、データやジャンプ先のアドレスです。

④2バイト命令（その2）

②と同様に2バイト命令ですが、両バイトともOPコードです。

⑤3バイト命令（その2）

③と同様に3バイト命令ですが、2バイトのOPコードと、1バイトのディスプレイスメントからなります。この

種の命令は、インデックスレジスタを用いる命令です。

ここで、ディスプレイスメント（displacement）とは「変位」を意味するもので、たとえば特定のアドレスを指定するとき、インデックスレジスタの値とのズレ（差分）を数値で入力します。

⑥４バイト命令（その１）

２バイトのOPコードと、２バイトのオペランドで構成されます。オペランドの下位バイト、上位バイトについての約束ごとは③と同じです。

⑦４バイト命令（その２）

２バイトのOPコード、ディスプレイスメント、オペランドの、以上４バイトで構成されます。この場合のオペランドは、イミディエイトデータです。

6.2　ダンプウィンドウを使う

Z-Visionには、ダンプウィンドウというものがあります。このデバッガを起動したとき、画面の左下に現れる横長のウィンドウです。ウィンドウが選択されているときは、黒いカーソル（▮）が点滅します。

これは、メモリの内容を表示したり編集（変更）したりするためのウィンドウで、16進数の数字ばかりが並びます。ウィンドウの左上にADRS（アドレスの意味）と記されていますが、その下方向には、0000Hから10Hずつ加算されたアドレス値が配置されています。

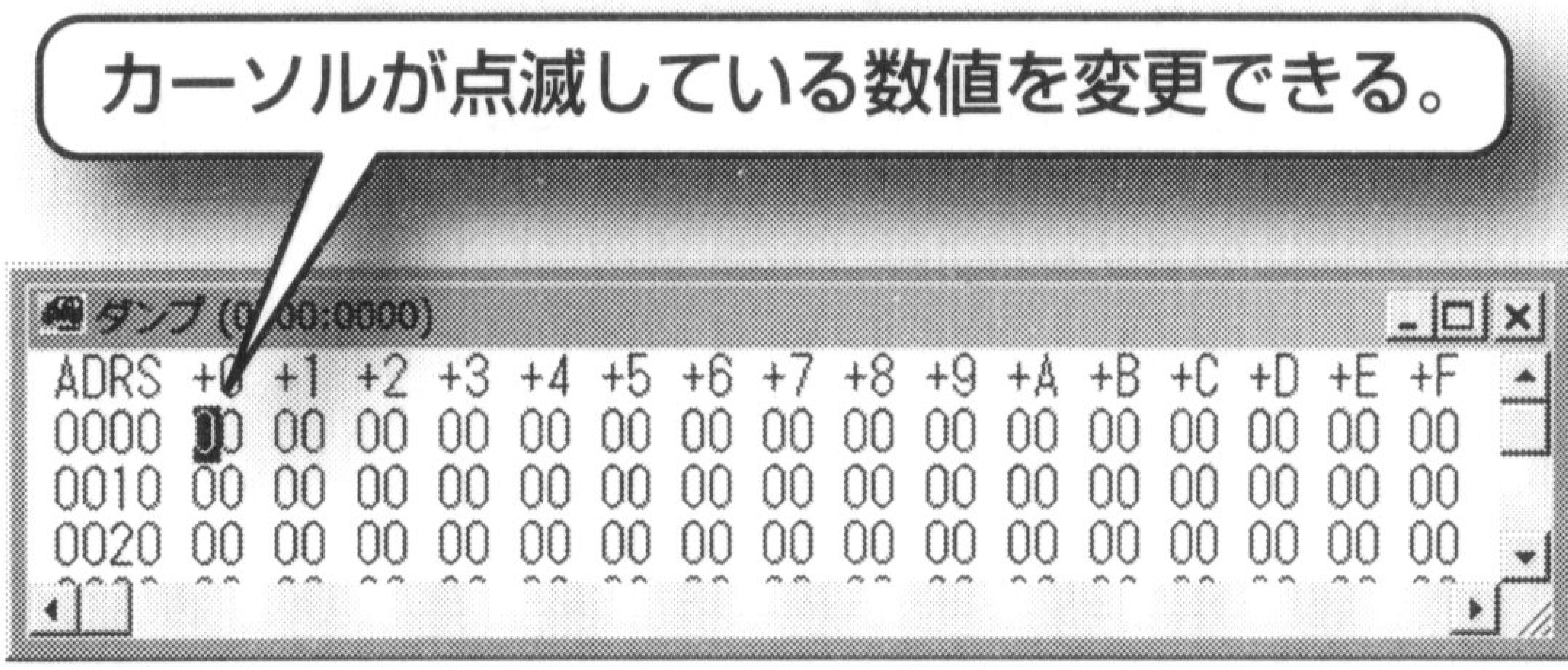


図6-2　ダンプウィンドウ

また、横方向はインクリメント、つまりアドレスが＋1ずつ加算された16個のデータが並ぶのです。逆アセンブルウィンドウが、命令文を基準にレイアウトされるのに対し、ダンプウィンドウではアドレスとデータがコンパクトに一覧できます。

実は、これら2つのウィンドウは常に連動しています。その様子を観察してみましょう。

カーソルの位置は、矢印キー（↑←↓→）を使って移動できますし、特定のデータ上でクリックすれば、直接移動することができます。

ここでは、0000H番地から、「C3 FF 55 62」と任意に**半角英数**でキーボードをたたいてみました。全角の文字は使えませんので注意してください。

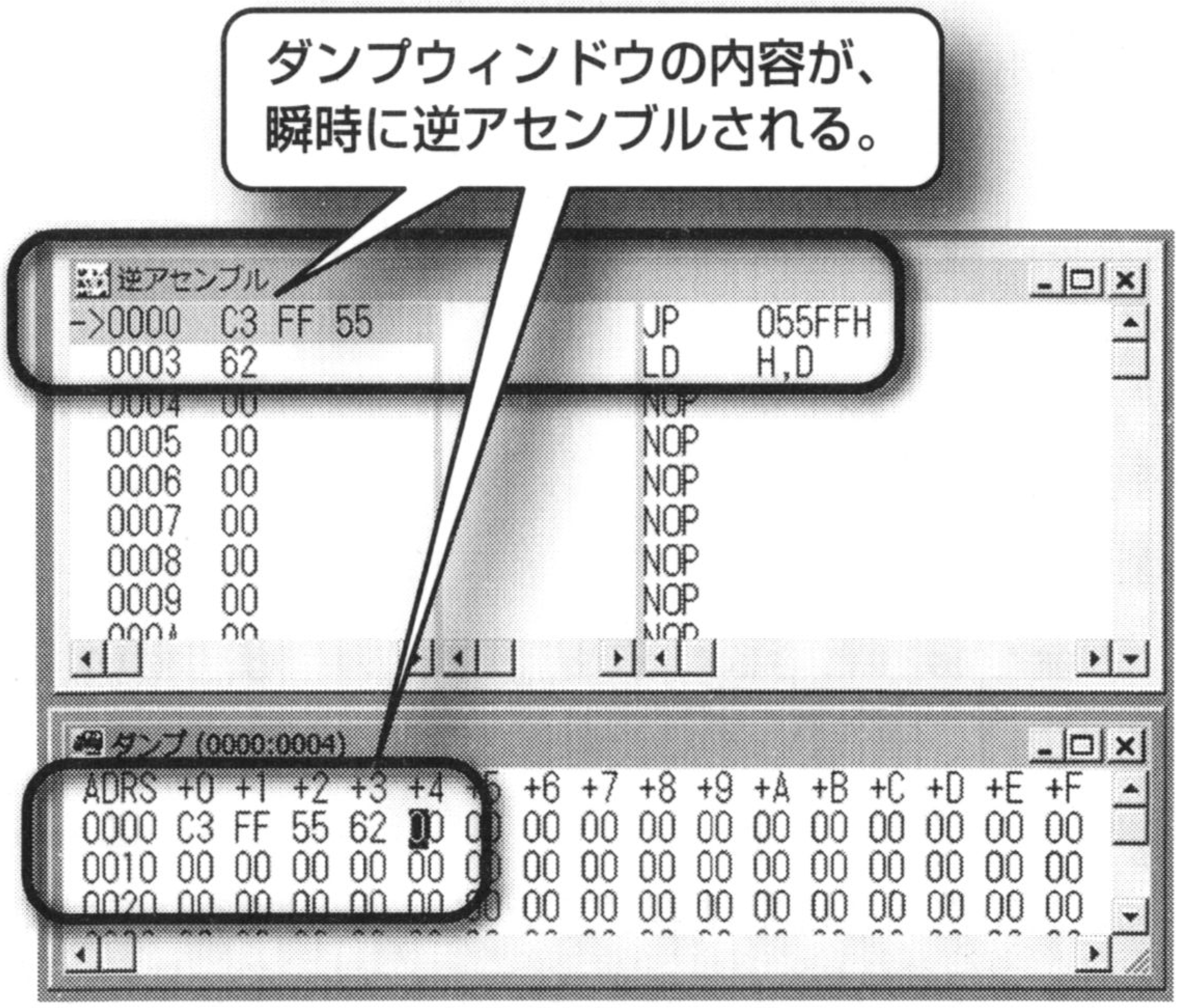


図6-3　任意に16進数を入力してみる

ここで入力した16進数は、たまたま思いついたもので、プログラムとしての意味を持ちません。しかし、ダンプウィンドウの内容が、すみやかに逆アセンブルウィンドウに反映されますので、CPUが何を認識して仕事をしているのかを垣間見ることができます。

また、この2つのウィンドウを常に同時に観察するよう心がけていると、しだいにアセンブリ言語の一部を機械語として記憶できるようになります。私たち人間がCPUに近づくのです。これは、アセンブリ言語がシンプルであると同時に、使用頻度の高い命令が限られているからです。

かつて、マイコンが日本に導入されたとき、「ハンドア

センブル」といって、手作業でアセンブル作業していたことを思い出します。筆者もその１人です。

ところで、ダンプウィンドウと逆アセンブルウィンドウの両方に共通していることですが、右端のスクロールバーを操作してみると、アドレス0000Hよりも上のアドレスを見ることができます。

すでにご理解いただいていると思いますが、アドレスFFFFHの次は0000Hであり、ループしています。たとえば、お金を扱うときに「FFFFH」円、つまり10進数で65535円に１円を足すと65536円です。

しかし、Z80のアドレスバスコントローラ（図4-6参照）は、16ビット幅だけのディジタル電子回路ですので、16ビットがすべて「１」のとき、インクリメント（＋１）すると、アドレス0000Hに戻るというわけです。

このアドレスの構造を知っておくと、あとで述べるコール/リターン命令やプッシュ/ポップ命令の仕組みが理解しやすくなります。

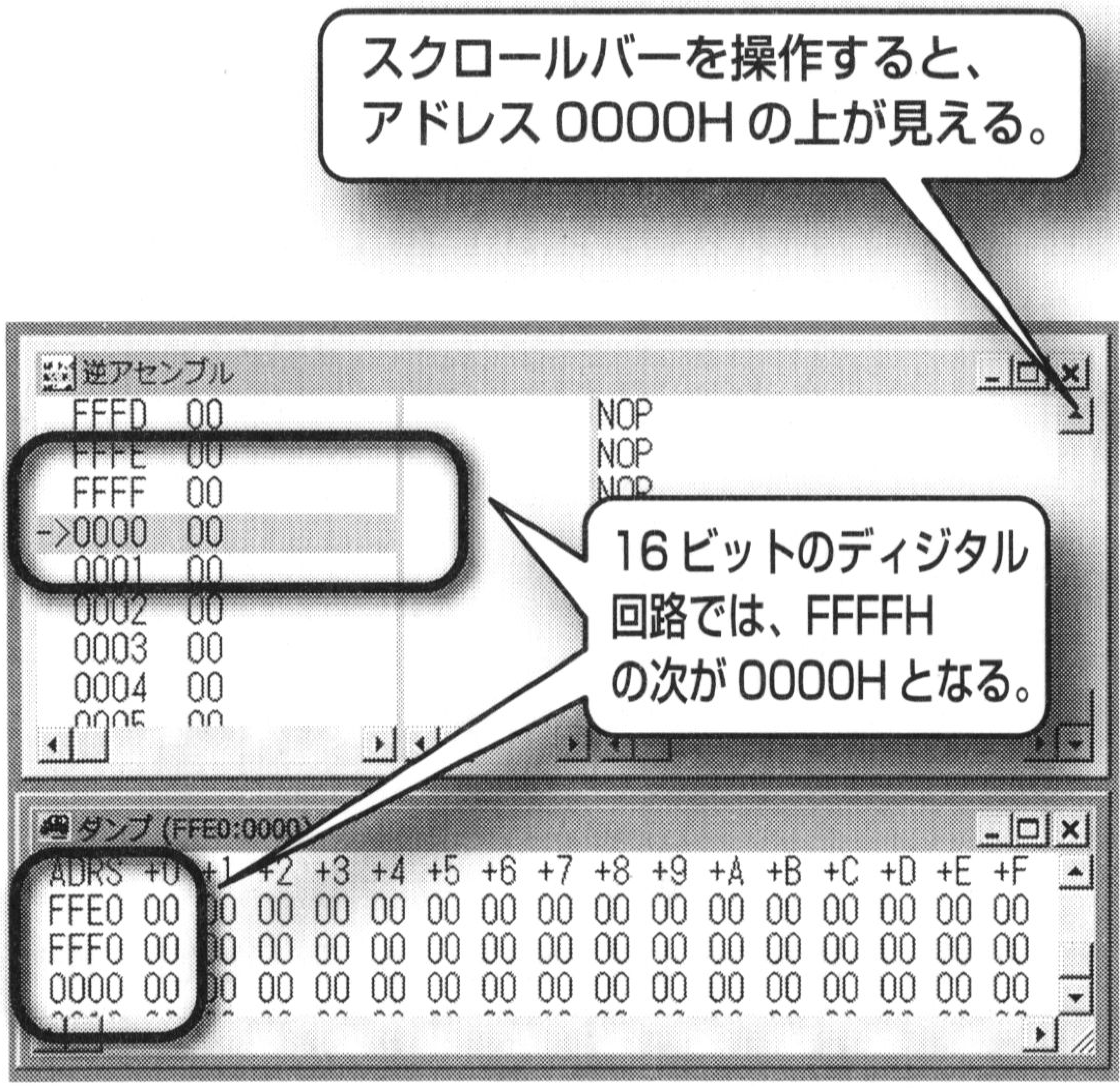


図6-4　アドレスFFFFHの次は0000H

ここまでの実験で、ダンプウィンドウを使えば、アセンブリ言語を使わずに、16進数で直接プログラムを入力できることがわかりました。

次の6.3節では、フラグの役割を理解しましょう。

6.3　フラグレジスタの機能

Z80には、単に演算を実行するだけでなく、演算結果の特徴をチェックして、その情報を記憶する機能があります。その場所が、フラグレジスタです。

図6-5は、Z80のフラグレジスタの様子を表したものです。8ビットのうちの4ビット（CY、Z、S、P/V）は、条件付きジャンプ命令により判定することができ、2ビットは10進演算専用の制御ビットです。残りの2ビットは未使用で、常に「0」です。

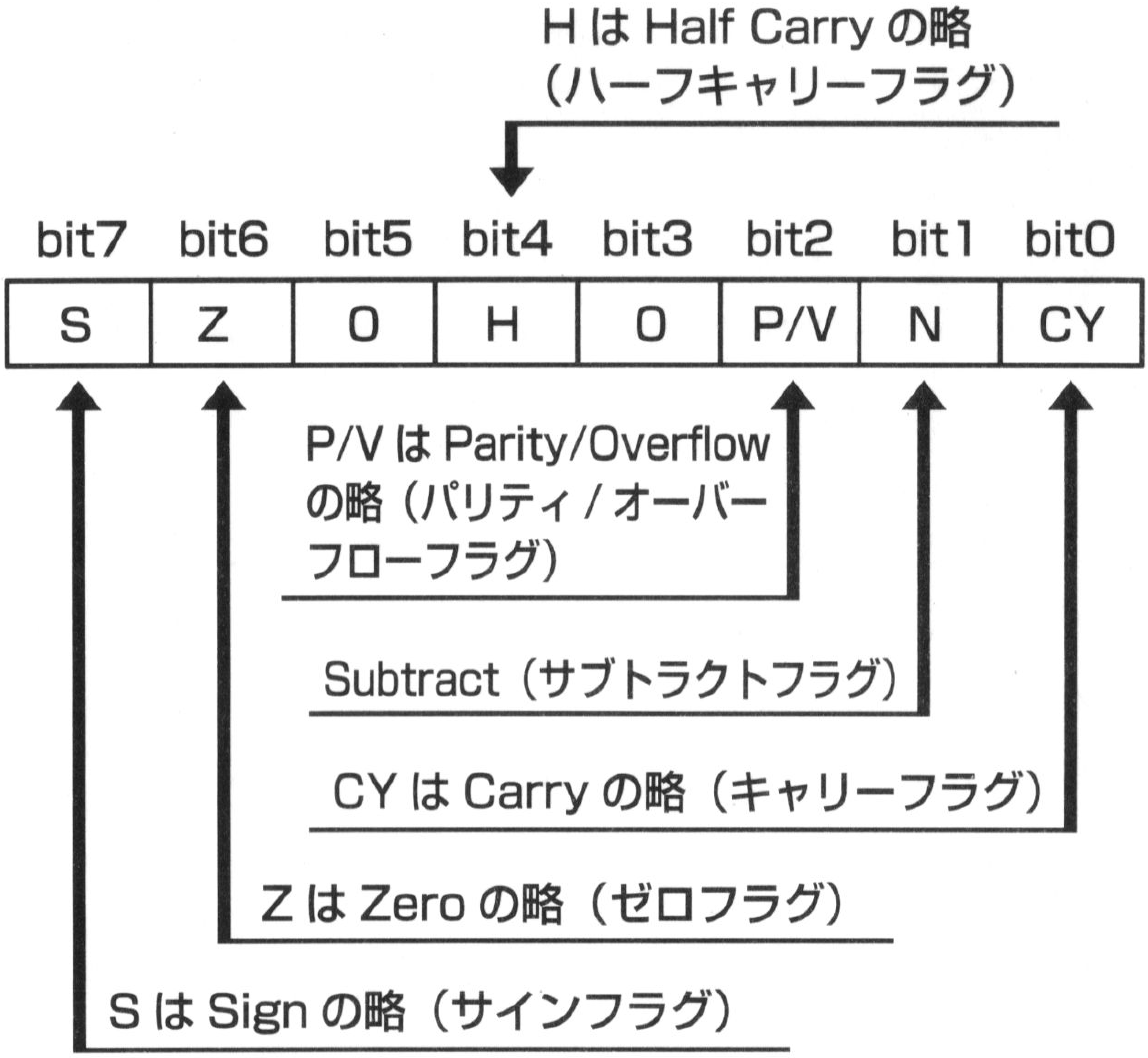


図6-5　フラグの種類

それぞれのフラグ（flag：旗）の役割を、命令の種類ごとに説明していきましょう。

■キャリーフラグ（CY）

キャリー（carry）とは、桁上げのことです。

①加算命令（ADD、ADC）

演算結果の最上位ビットから桁上げを生じたときにセットされます。ここで言うセット（set）とは、値を「1」に設定することです。

②減算命令（SUB、SBC、NEG、CP）

上位桁から借り（ボロー）を生じたときにセットされます。

③ローテート/シフト命令（RL、RLC、RRなど）

最上位ビット（左シフトのとき）、または最下位ビット（右シフトのとき）の状態がキャリーフラグにセットされます。

④キャリーフラグの直接コントロール

SCF命令を実行すると、キャリーフラグは強制的に「1」にセットされ、CCF命令を実行するとキャリーフラグの状態は反転（0→1、1→0）します。

⑤論理演算命令（AND、OR、XOR）

これらを実行すると、キャリーフラグは必ず「0」になります。

■ゼロフラグ（Z）

演算結果がゼロであることを示すフラグです。

①加算命令、減算命令、論理演算命令、インクリメント/デクリメント命令（INC、DEC）

演算結果がゼロのときに、ゼロフラグは「1」になります。

②ローテート/シフト命令

演算の結果によって、ゼロフラグが変化します。

③入力命令

IN A,(n)命令ではゼロフラグは変化しませんが、IN r,(C)命令では、読み込まれたデータにしたがって、ゼロフラグが変化します。

④ビットテスト命令

BIT b,s命令では、指定したビットに応じてゼロフラグが変化します。

⑤その他

Aレジスタの値を反転するCPL命令では、ゼロフラグは変化しません。

■サインフラグ（S）

サインフラグは、演算結果の符号を示します。演算結果が正（プラス）であれば、サインフラグは「0」になり、負（マイナス）であればサインフラグは「1」になります。

①加算命令、減算命令

サインフラグは、基本的には加算命令（ADD、ADC）および減算命令（SUB、SBC、CP、NEG）の結果の符号ビットの状態を示します。

「符号ビット」とは、符号付き整数の正と負を区別するための表現方法です。図6-6と6-7は、それぞれ8ビットと16ビットについて符号付き整数を一覧にしたものです。

符号付き整数の表現では、最上位ビットは符号ビットとして取り扱われ、正の場合には「0」であり、負の場合には「1」となります。

8ビットのデータでは、－128（80H）から＋127（7FH）の範囲を表現でき、16ビットの場合には、－32768（8000H）から＋32767（7FFFH）の範囲を表現できます。

②その他

算術演算以外にも、論理演算命令（AND、OR、XOR）やINC/DEC命令、ローテート/シフト命令、IN r,(C)命令などで、演算結果の符号ビットの状態がセットされます。

10 進数	16 進数	2 進数	
127	7F	0	1111111
126	7E	0	1111110
125	7D	0	1111101
⋮	⋮		⋮
2	02	0	0000010
1	01	0	0000001
0	00	0	0000000
−1	FF	1	1111111
−2	FE	1	1111110
⋮	⋮		⋮
−126	82	1	0000010
−127	81	1	0000001
−128	80	0	0000000

最上位ビット（符号ビット）

図6-6　符号付き整数（8ビット）

10進数	16進数	2進数
32767	7FFF	0 1111111 11111111
32766	7FFE	0 1111111 11111110
32765	7FFD	0 1111111 11111101
2	0002	0 0000000 00000010
1	0001	0 0000000 00000001
0	0000	0 0000000 00000000
−1	FFFF	1 1111111 11111111
−2	FFFE	1 1111111 11111110
−32766	8002	1 0000000 00000010
−32767	8001	1 0000000 00000001
−32768	8000	0 0000000 00000000

最上位ビット（符号ビット）

図6-7　符号付き整数（16ビット）

■パリティ/オーバーフローフラグ（P/V）

２つの目的を持つフラグです。

①パリティフラグ

論理演算（AND A,Bなど）を実行する場合は、その結果のパリティ極性を示します。

「パリティ極性」とは、演算の結果、２進数８ビットの値の中で、「１」の数が偶数個あるか奇数個かを意味します。偶数ならば「１」、奇数ならば「０」です。

論理演算命令（AND、OR、XOR）やローテート/シフト命令、DAA命令、IN r,(C)命令などが関係します。

②オーバーフローフラグ

加算命令（ADD、ADC）や減算命令（SUB、SBC）で、符号付きの「2の補数」による算術演算を実行する場合は、オーバーフローしたかどうかを示します。演算結果がオーバーフローすれば「１」になります。

「２の補数」とは、２進法で表記された「ある数値」に足し算することによって、全体の桁が１つ上がるような最小の数値のことです。

ここでは、具体的な求め方を示した方がよいでしょう。

２進法には「１の補数」と「２の補数」があって、１の補数とは、単純に「１」と「０」を入れ替えた数値を指します。２の補数は、「１」の補数に１を加算したものです。たとえば、２進数101011110の１の補数は010100001ですから、２の補数は010100001＋1＝

010100010となります。

■ハーフキャリーフラグ（H）

このフラグは、加算命令（ADD、ADC）、減算命令（SUB、SBC、CP、NEG）、INC命令、DEC命令を実行したときに、下位４ビットから上位４ビットに対して、桁上がりまたは桁借りがあれば、「１」にセットされます。

ただし、条件付きジャンプ命令を用いて判定することはできません。

特徴的な機能としては、10進演算における10進補正命令（DAA）を実行すると、このフラグとキャリーフラグの状態にしたがって、巧みに10進補正を行います。

これが、２進化10進数（BCD：binary-coded decimal）です。

２進化10進数は、コンピュータにおける数値表現の１つで、10進表現の１桁を、０から９までを表す４桁の２進数で表したものです。

一般に４桁の２進数は、０から15までの整数を表すことができますが、２進化10進数では、このうち最初の10個を有効な数値として扱います。

つまり、０から９までの数を、10進数とBCDで表現すると、図6-8のようになります。このルールにしたがうと、たとえば、10進数の「123」は0001 0010 0011と表記されます。

10進数	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

図6-8　10進数とBCD

■サブトラクトフラグ（N）

サブトラクトは「減算」の意味です。

直前に実行された命令が、加算命令（ADD、ADC、INC）か減算命令（SUB、SBC、CP、NEG、DEC）かを示すためのフラグです。

加算命令が実行されると「0」、減算命令が実行されると「1」になります。10進補正命令（DAA）のためのフラグです。

なお、このフラグによる条件付きジャンプ命令の判定はありません。

第7章　さまざまな命令語をためす

Z80の命令語は、大きく次の８つのグループに分けられます。

①ロード、イクスチェンジ
②入出力
③ジャンプ、コール、リターン
④算術演算、論理演算
⑤ローテート、シフト
⑥ビット操作
⑦CPU制御
⑧ブロック転送、ブロックサーチ

本書では、これらの中からマイコン入門者に適当と思われるものだけを選定しました。そのため、インデックスレジスタや割り込み処理、ブロック転送などに関連する命令語については、あえて割愛しました。

7.1　主な命令

ここから、Z-Visionを用いたCPUの動作実験に適していると思われる命令を紹介していきます。

プログラムを作るときには、データの変化に応じて、異なる命令を実行するといった分岐が欠かせません。そこで、各命令の動きを説明する表には、図7-1に示すように、記号を用いたフラグ変化表を個別に掲載しています。

なお、6つのフラグのいずれもが変化しない場合は、「◆フラグは変化しない。」などと記載しています。

◆	フラグは変化しない。
×	フラグの状態が不定で、0にも1にもなる。
0	フラグが0になる。
1	フラグが1になる。
↑↓	演算結果に影響されて、0または1になる。

図7-1　フラグの変化の種類

7.2　CPU制御命令

CPU制御命令には、図7-2に示すHALT命令とNOP命令のほか、本書では割愛している割り込み処理に関連する命令が含まれます。

たとえば、プログラムを1回だけ実行して終了するときは、最後の行にHALTを入れます。これは、CPUの強制停止命令で、機械語は76Hです。

あとに出てくるシミュレート例でも、多くの場合、このHALTを使っています。これがないと、プログラムが暴走してしまうからです。暴走とは、ユーザが想定していないアドレスの命令を勝手に実行することを指します。

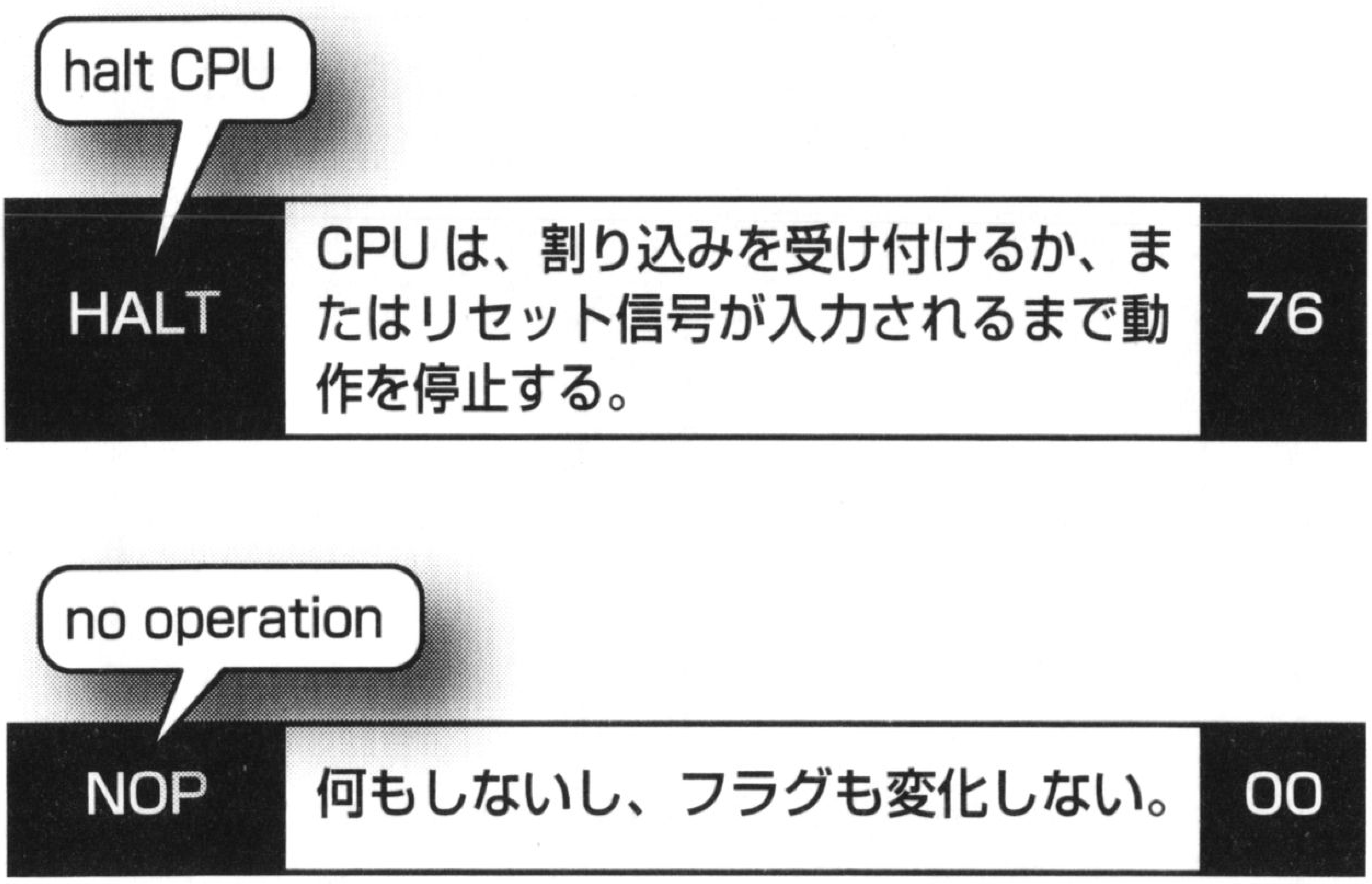


図7-2　HALT命令とNOP命令

また、初めてZ-Visionを起動してみて、逆アセンブルウィンドウにNOPがたくさん並んでいることに疑問を感じた読者がいると思います。

NOPは、「no operation」の略語です。Z-Visionを起動したときは、プログラムがロードされておらず、メモリの値がすべて0になっているため、逆アセンブルウィンドウは、単純にそれを拾ってニーモニックを表示しているのです。特別な意味はありません。

なお実際のマイコンシステムでは、わずかな時間のズレをつくることによって、電子回路が安定に動作することがあって、その調節にNOPを利用することがあります。

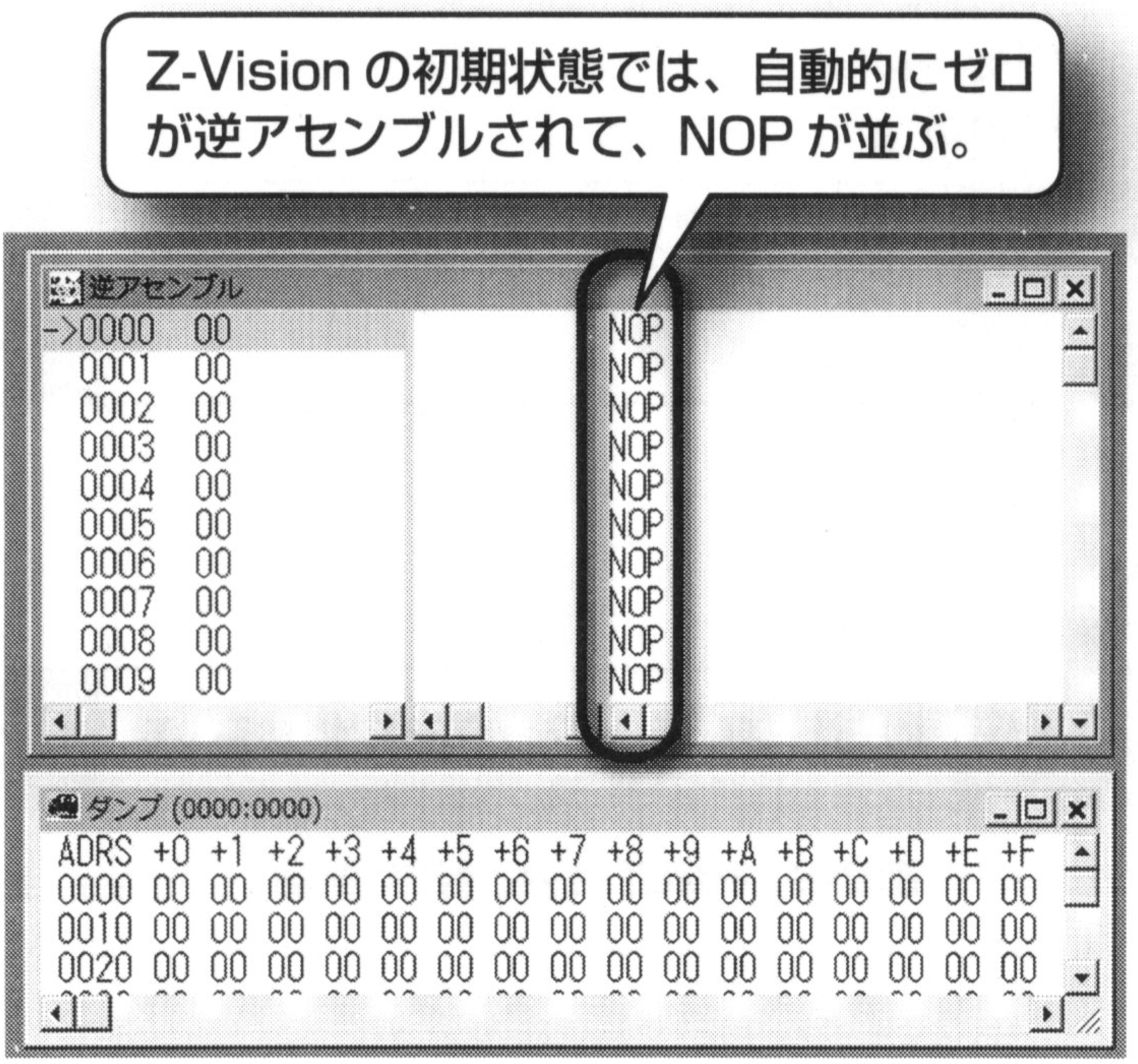


図7-3　データのない逆アセンブルウィンドウ

7.3　ロード命令

ロード命令は、データをCPU内部のレジスタ間、あるいはレジスタとメモリの間で転送するものです。

この命令には、転送されるべきデータの出発点（ソース：source）と到着点（ディスティネーション：destination）が指定されます。そして、データの出発点のデータは変更されないという特徴があります。つまり、「転送」とはデータのコピーを意味します。

ロード命令のニーモニックは、たとえば、1バイトの数値nをHLレジスタペアが示すメモリ内へ移すときは、LD (HL),nとなります。

ここで、HLをかこむカッコ（　）は、HLレジスタペアの値がメモリ内の場所を指すアドレスとして使われることを意味します。言い換えると、HLレジスタペアが示すメモリの内容を表すときに、(HL)と表記するのです。

また、ロード命令をニーモニックで記述する場合、到着点が必ず最初に置かれ、出発点はうしろに置かれる点に注意してください。

それでは、図7-4の命令を、ダンプウィンドウで機械語入力し、実行してみましょう。0000H番地から、「21 10 00 36 FF 76」と入力します。

この場合、あらかじめHLに、メモリのアドレスを入力しておかなくてはなりません。図7-13の16ビットロード命令も必要です。

図7-5～図7-8では、HLを0010H、ロードする数値nはFFHとしました。

LD (HL),n	1バイトの値nを、HLレジスタペアが示すアドレスのメモリへロードする。　◆フラグは変化しない。	36 n

load location (HL) with value n

図7-4　8ビットLD命令（1）

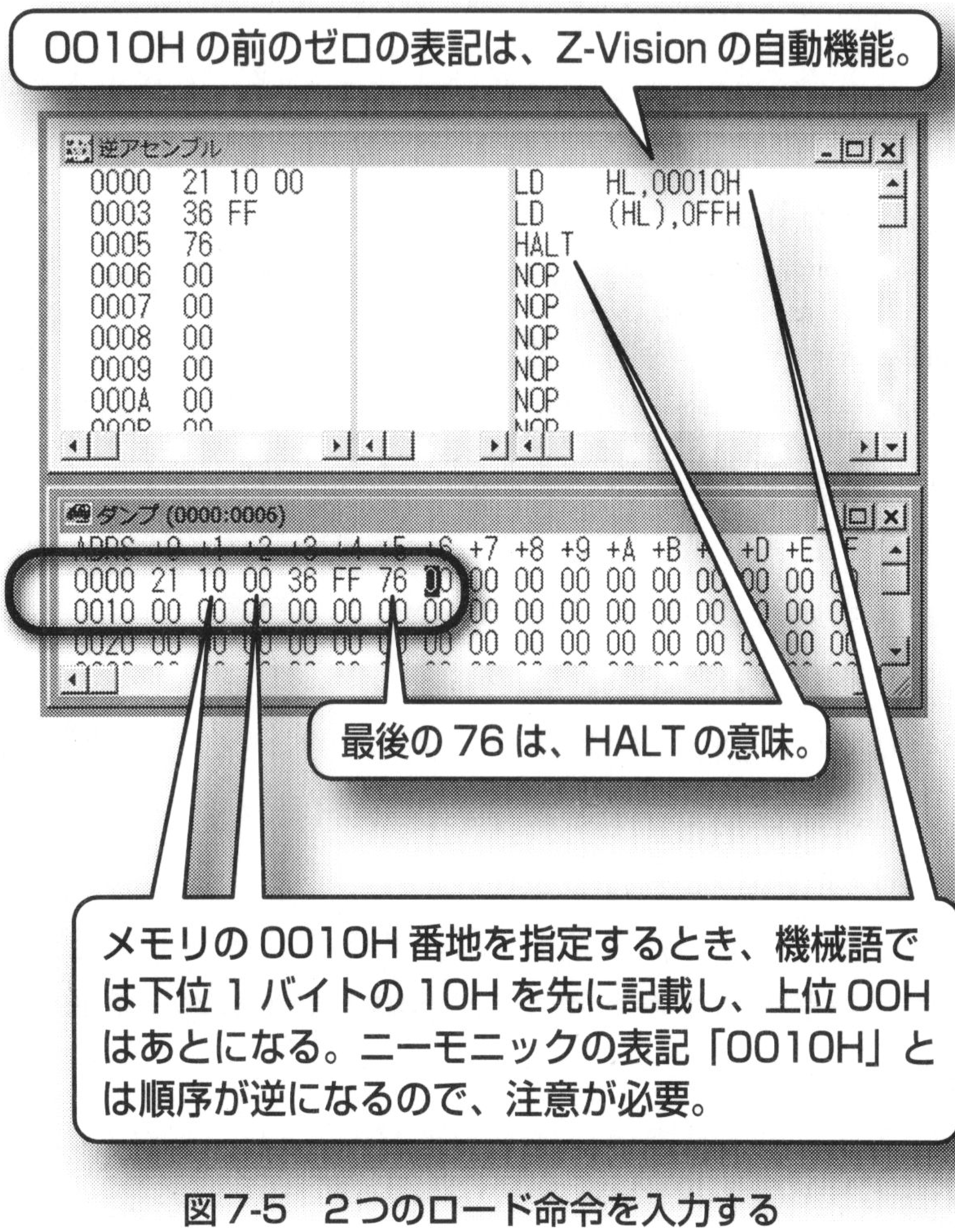
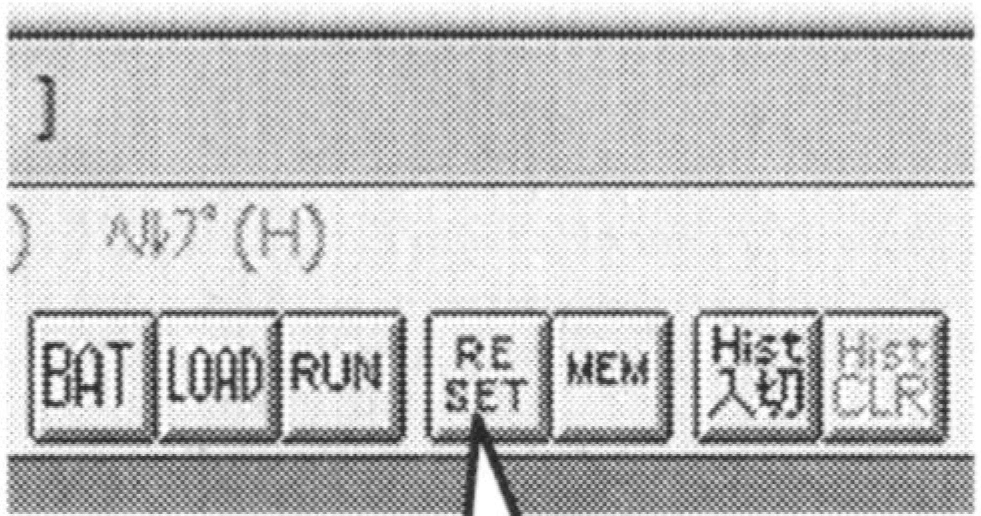


図7-5　2つのロード命令を入力する



シミュレータで使う16進数は、ふつう4桁までです。10進数で考える数値を16進数に置き換えるのは、めんどうなようですが、以下の数字の規則（1⇒2⇒4⇒8と2倍ずつ増える）を覚えていると、暗算に便利です。

01H=1D	10H= 16D	0100H= 256D
02H=2D	20H= 32D	0200H= 512D
04H=4D	40H= 64D	0400H=1024D
08H=8D	80H=128D	0800H=2048D

図7-6　レジスタをリセットする

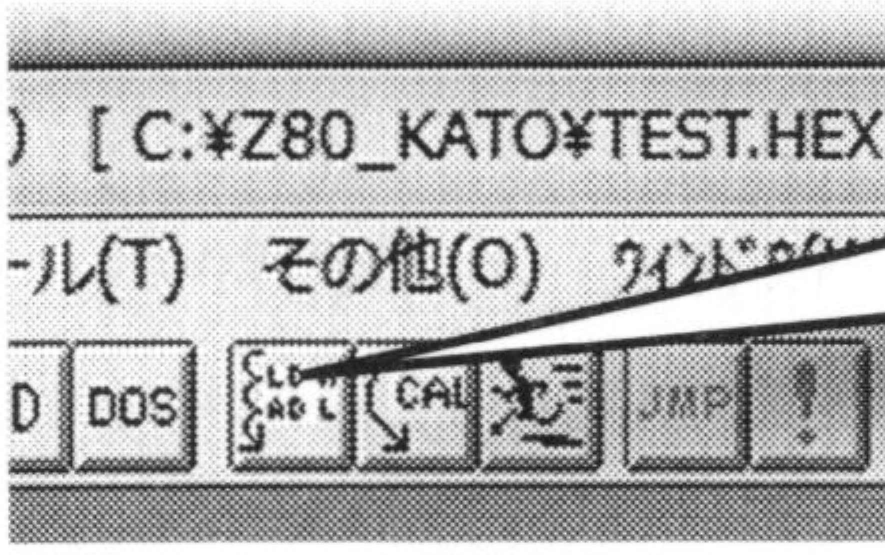


図7-7　「1行トレース」ボタンを2回クリック

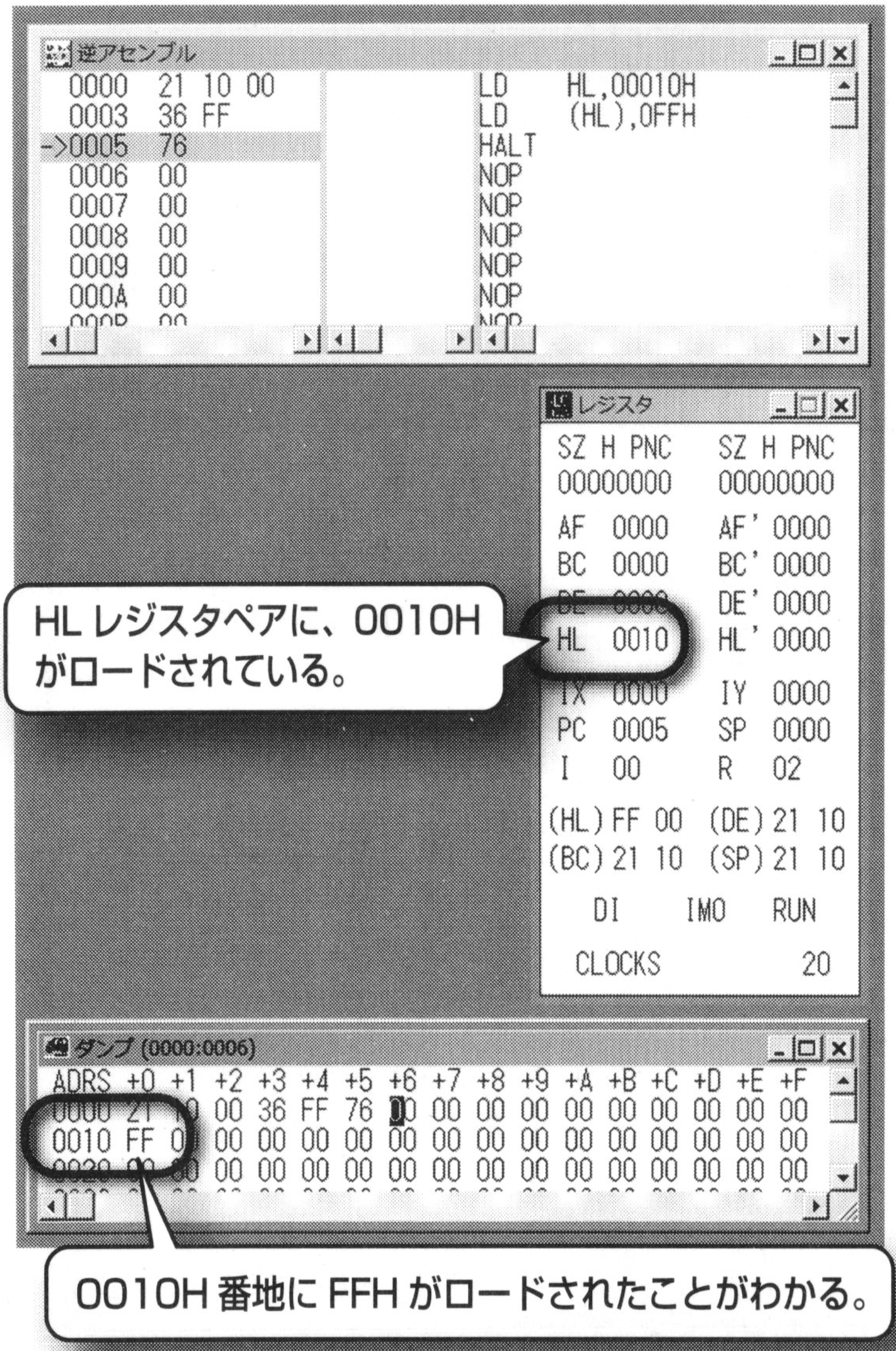

逆アセンブル
0000 21 10 00 LD HL,00010H
0003 36 FF LD (HL),0FFH
->0005 76 HALT
0006 00 NOP
0007 00 NOP
0008 00 NOP
0009 00 NOP
000A 00 NOP
レジスタ
SZ H PNC SZ H PNC
00000000 00000000
AF 0000 AF' 0000
BC 0000 BC' 0000
DE' 0000
HL 0010 HL' 0000
IY 0000
PC 0005 SP 0000
I 00 R 02
(HL) FF 00 (DE) 21 10
(BC) 21 10 (SP) 21 10
DI IMO RUN
CLOCKS 20
HLレジスタペアに、0010Hがロードされている。
ダンプ (0000:0006)
ADRS +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0010 FF
0010H番地にFFHがロードされたことがわかる。

図7-8　実行

load location (HL) with register r

LD (HL),r

r レジスタの値を、HL レジスタペアが示すアドレスのメモリへロードする。
◆フラグは変化しない。

r→	A	B	C	D	E	H	L
機械語	77	70	71	72	73	74	75

LD (ss),A

A レジスタの値を、ss レジスタペアが示すアドレスのメモリへロードする。
◆フラグは変化しない。

ss→	BC	DE
機械語	02	12

load location register pair (ss) with accumulator

LD (nn),A

A レジスタの値を、2 バイト値 nn で示されるアドレスのメモリにロードする。◆フラグは変化しない。

32
nn

load location (nn) with accumulator

図7-9　8ビットLD命令（2）

LD A,(ss)

ssレジスタペアが示すアドレスの内容を、Aレジスタへロードする。

◆フラグは変化しない。

ss→	BC	DE
機械語	0A	1A

load accumulator with location register pair (ss)

load register r with location (HL)

LD r,(HL)

HLレジスタペアが示すアドレスの内容を、rレジスタへロードする。

◆フラグは変化しない。

r→	A	B	C	D	E	H	L
機械語	7E	46	4E	56	5E	66	6E

LD A,(nn)

2バイト値nnで示されるアドレスの内容を、Aレジスタにロードする。

◆フラグは変化しない。

3A
nn

load accumulator with location (nn)

図7-10　8ビットLD命令（3）

load register r with register r’

LD r,r’

r’レジスタの内容を、r レジスタへロードする。　◆フラグは変化しない。

注：ここでの r’は補助レジスタの意味ではなく、いずれも主レジスタです。

r＼r’	A	B	C	D	E	H	L
A	7F	78	79	7A	7B	7C	7D
B	47	40	41	42	43	44	45
C	4F	48	49	4A	4B	4C	4D
D	57	50	51	52	53	54	55
E	5F	58	59	5A	5B	5C	5D
H	67	60	61	62	63	64	65
L	6F	68	69	6A	6B	6C	6D

load register r with value n

LD r,n

1バイト値 n を、r レジスタへロードする。
◆フラグは変化しない。

r→	A	B	C	D	E	H	L
機械語	3E n	06 n	0E n	16 n	1E n	26 n	2E n

図7-11　8ビットLD命令（4）

load location (nn) with register pair ss

LD (nn),ss

ss レジスタペアの値を、2 バイト値 nn が示すアドレスのメモリへロードする。ss の下位 1 バイトは nn 番地に、ss の上位 1 バイトは nn+1 番地にロードされる。

◆フラグは変化しない。

ss→	BC	DE	HL	SP
機械語	ED 43 nn	ED 53 nn	ED 63 nn	ED 73 nn

LD (nn),HL

HL レジスタペアの値を、2 バイト値 nn が示すアドレスにロードする。L レジスタの値は nn 番地に、H レジスタの値は nn+1 番地にロードされる。

◆フラグは変化しない。

22
nn

load location (nn) with HL

図 7-12　16 ビット LD 命令（1）

load register pair ss with location (nn)

LD ss,(nn)

2バイト値nnが示すアドレスの内容を、ssレジスタペアへロードする。nn番地の値はssの下位1バイトに、nn+1番地の値はssの上位1バイトにロードされる。
◆フラグは変化しない。

ss→	BC	DE	HL	SP
機械語	ED 4B nn	ED 5B nn	ED 6B nn	ED 7B nn

load register pair ss with value nn

LD ss,nn

2バイト値nnをssレジスタペアへロードする。nn番地の値はssの下位1バイトに、nn+1番地の値はssの上位1バイトにロードされる。　◆フラグは変化しない。

ss→	BC	DE	HL	SP
機械語	01 nn	11 nn	21 nn	31 nn

LD HL,(nn)

2バイト値nnが示すアドレスの内容を、HLレジスタペアへロードする。nn番地の値はLレジスタに、nn+1番地の値はHレジスタにロードされる。　◆フラグは変化しない。

2A nn

load HL with location (nn)

図7-13　16ビットLD命令（2）

7.4　イクスチェンジ命令

ロード命令とイクスチェンジ命令については、第５章のテストプログラムを実行する場面で、その動きを確認しました。実は、主レジスタと補助レジスタのデータの交換を行うイクスチェンジ命令も、ロード命令の一種です。

EX AF,AF’	A レジスタと F レジスタの表（主レジスタ）と裏（補助レジスタ）の各内容を交換する。 ◆フラグは変化しない。	08

exchange the contents of AF and AF’

EX DE,HL	DE レジスタペアと HL レジスタペアの内容を交換する。 ◆フラグは変化しない。	EB

exchange the contents of DE and HL’

EXX	B、C、D、E、H、L の各レジスタの表と裏の各内容を一度に交換する。 ◆フラグは変化しない。	D9

exchange the register bank

図7-14　EX命令とEXX命令

7.5　入出力命令

Z80は、入出力命令によって、汎用レジスタとI/Oポートの間でデータの転送を行うことができます。

I/Oのポート番号は、アドレスバスの下位8ビットによって示され、I/Oアドレスと呼ばれます。

多くの入出力命令では、ポート番号を命令コードの第2バイトによって指定しますが、これをCレジスタの内容によって指定する命令もあります。

■I/Oウィンドウの活用

I/OポートはCPUとは離れたところにあって、メモリとの関係に似ています。Z-Visionには、これをシミュレートする機能があって、それがI/Oウィンドウです。これは、指定したI/Oアドレスの内容を常に表示します。

図7-16～図7-23は、簡単な入出力命令の実験をする様子です。

ここでは、入力ポートの内容をAレジスタに取り込むプログラムと、逆にAレジスタの値を出力ポートに送るプログラムを続けて紹介します。

IN r,(C)

Cレジスタが示す入力ポートの内容を、rレジスタに入力する。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	◆

r→	A	B	C	D	E	H	L
機械語	ED 78	ED 40	ED 48	ED 50	ED 58	ED 60	ED 68

load the register r with input from port (C)

パリティフラグ

IN A,(n)

入力ポートnの内容を、Aレジスタに入力する。　◆フラグは変化しない。

DB n

load the accumulator with input from port (n)

OUT(C),r

rレジスタの内容を、Cレジスタが示す出力ポートへ出力する。◆フラグは変化しない。

r→	A	B	C	D	E	H	L
機械語	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

load output port (C) with r

OUT(n),A

Aレジスタの内容を、出力ポートnへ出力する。　◆フラグは変化しない。

D3 n

load output port (n) with accumulator

図7-15　入出力命令IN命令とOUT命令

①ツールバーの「I/Oデータの表示」ボタンを押すか、ビューメニューから「I/O」を選ぶと、I/Oウィンドウが開きます。

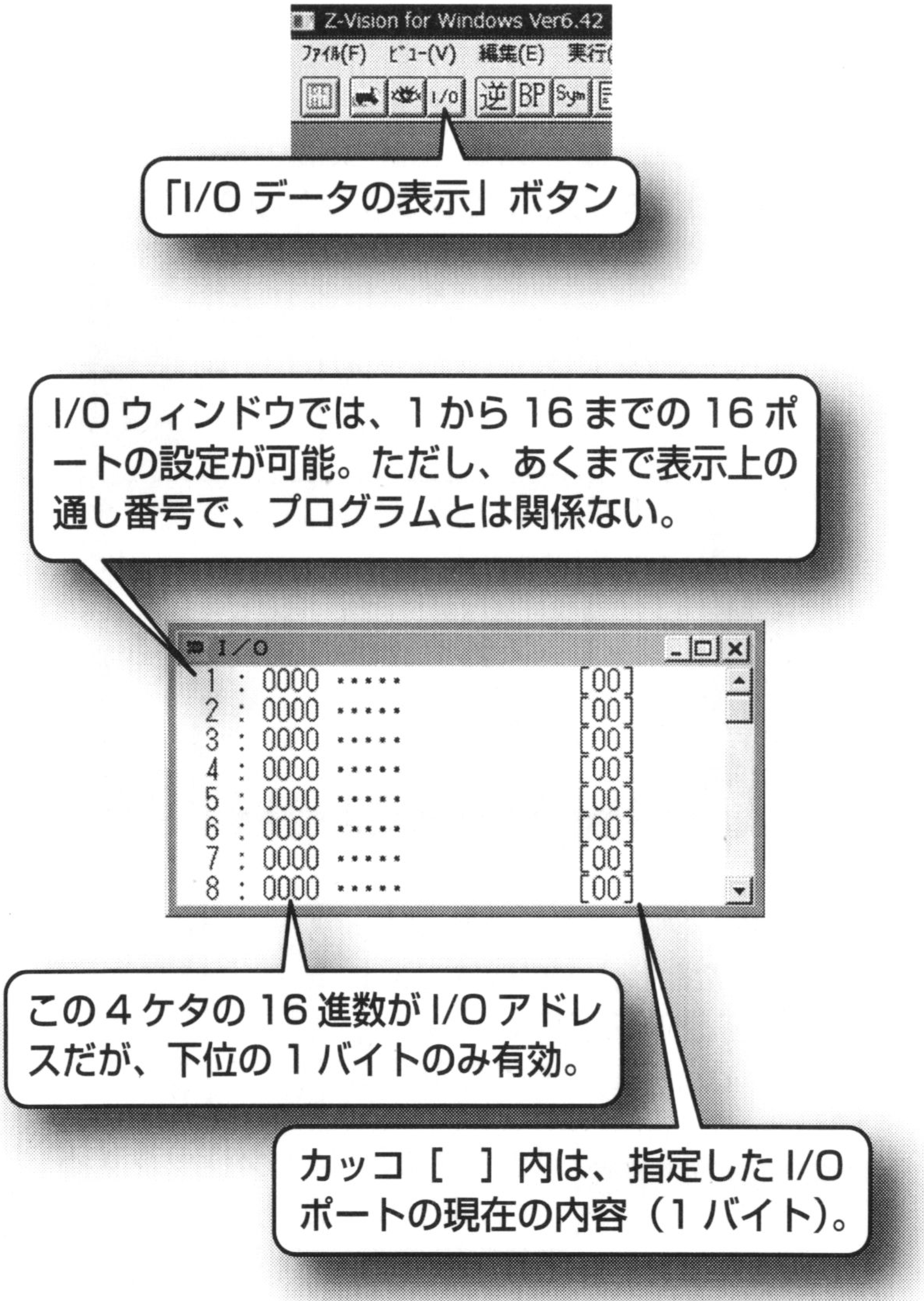


図7-16　I/Oウィンドウを開く

②ここでは、最初のNo.1をI/OアドレスAAHの出力ポートに設定してみましょう。No.1の行の上でダブルクリックすると、「I/Oの設定」ダイアログボックスが現れますので、「ポート番号」欄に0AAHと入力し、「追加」ボタンをクリックします。

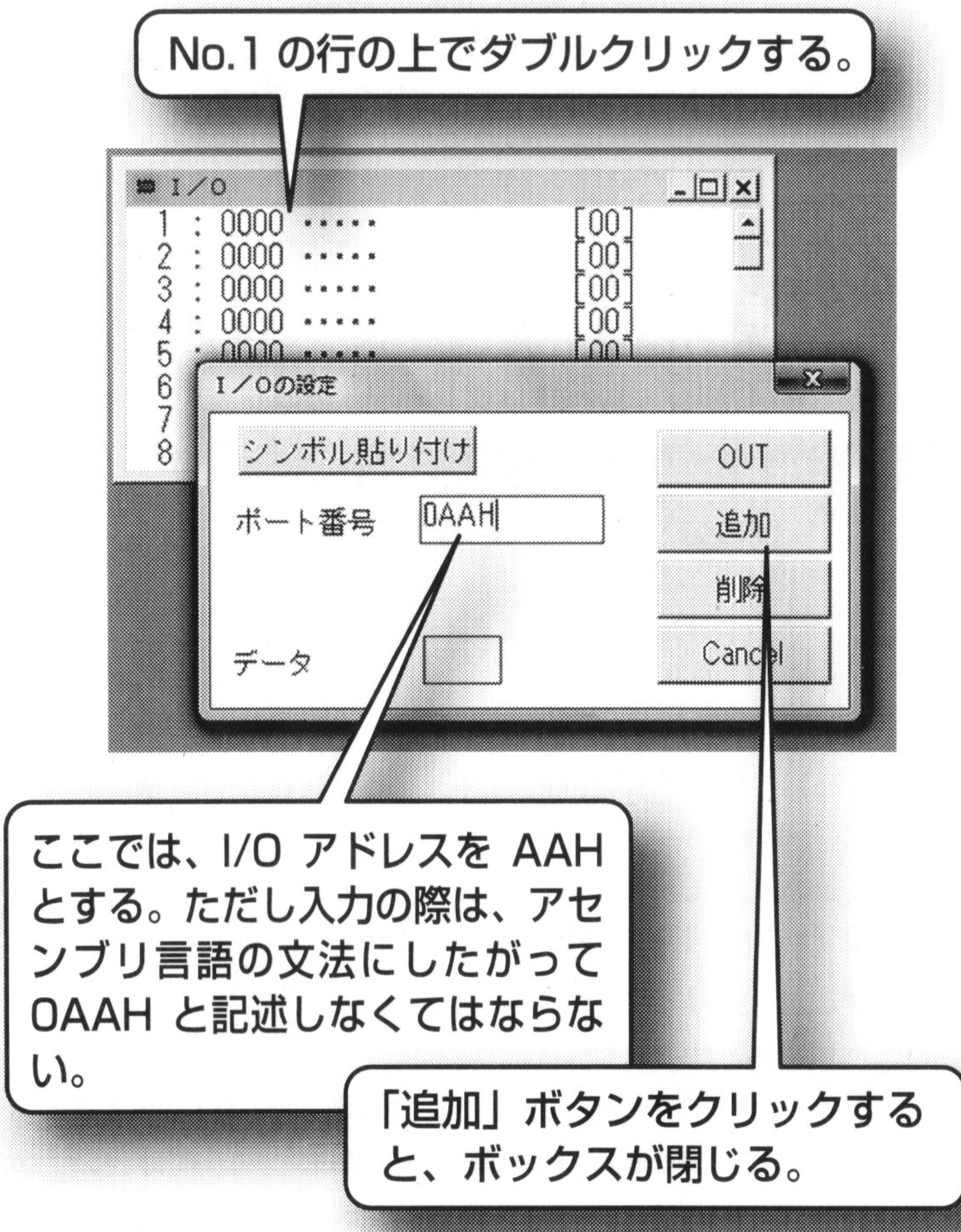


図7-17　No.1を出力ポートに設定する

③ダンプウィンドウで、「3E 55 D3 AA 76」を入力します。これは、55HというデータをAAHというI/Oポートに出力するプログラムです。

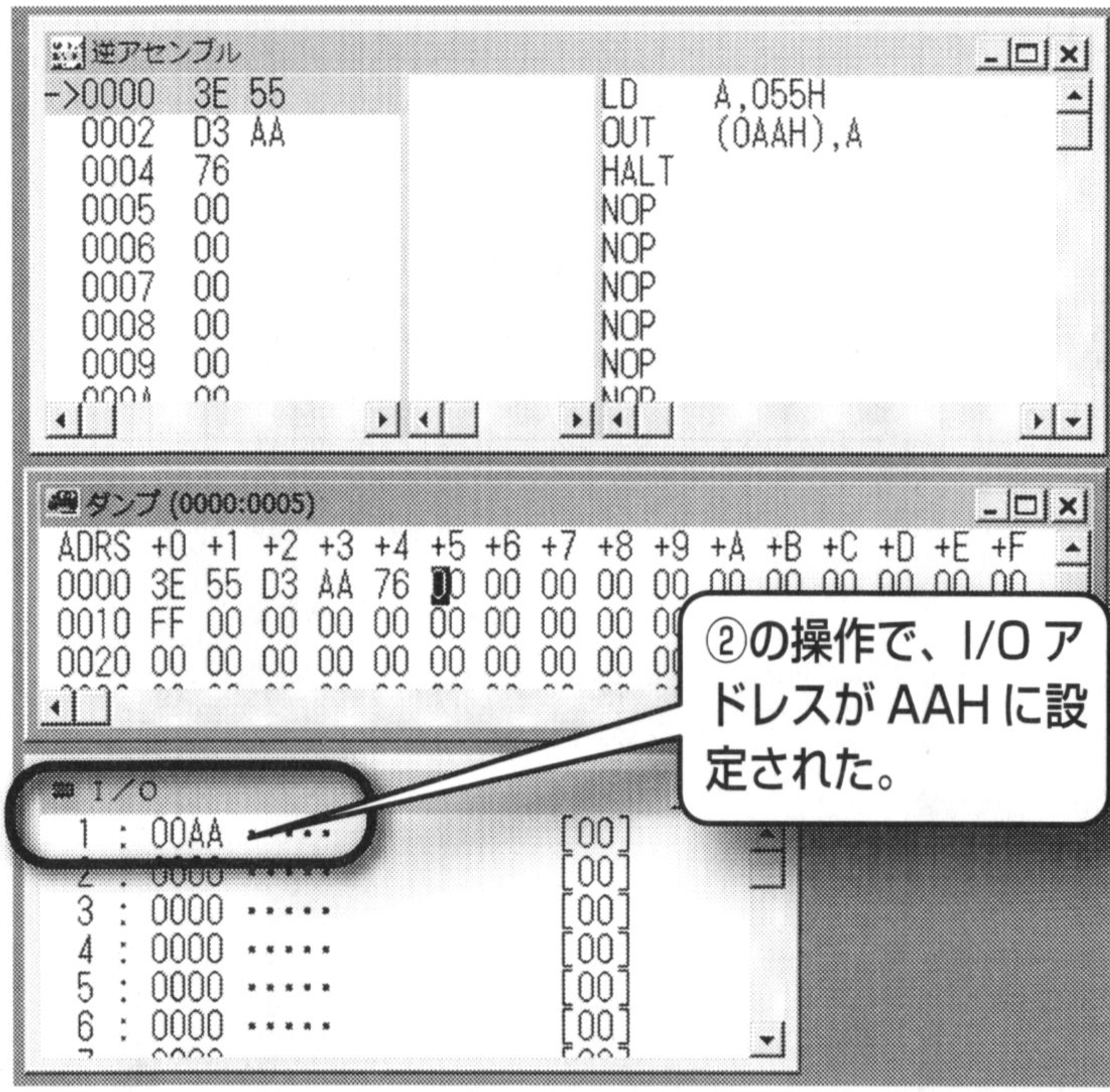


図7-18　出力プログラムを入力する

④③のプログラムを実行すると、I/OウィンドウのNo.1のデータが55Hに変更されます。データが出力されたという意味です。

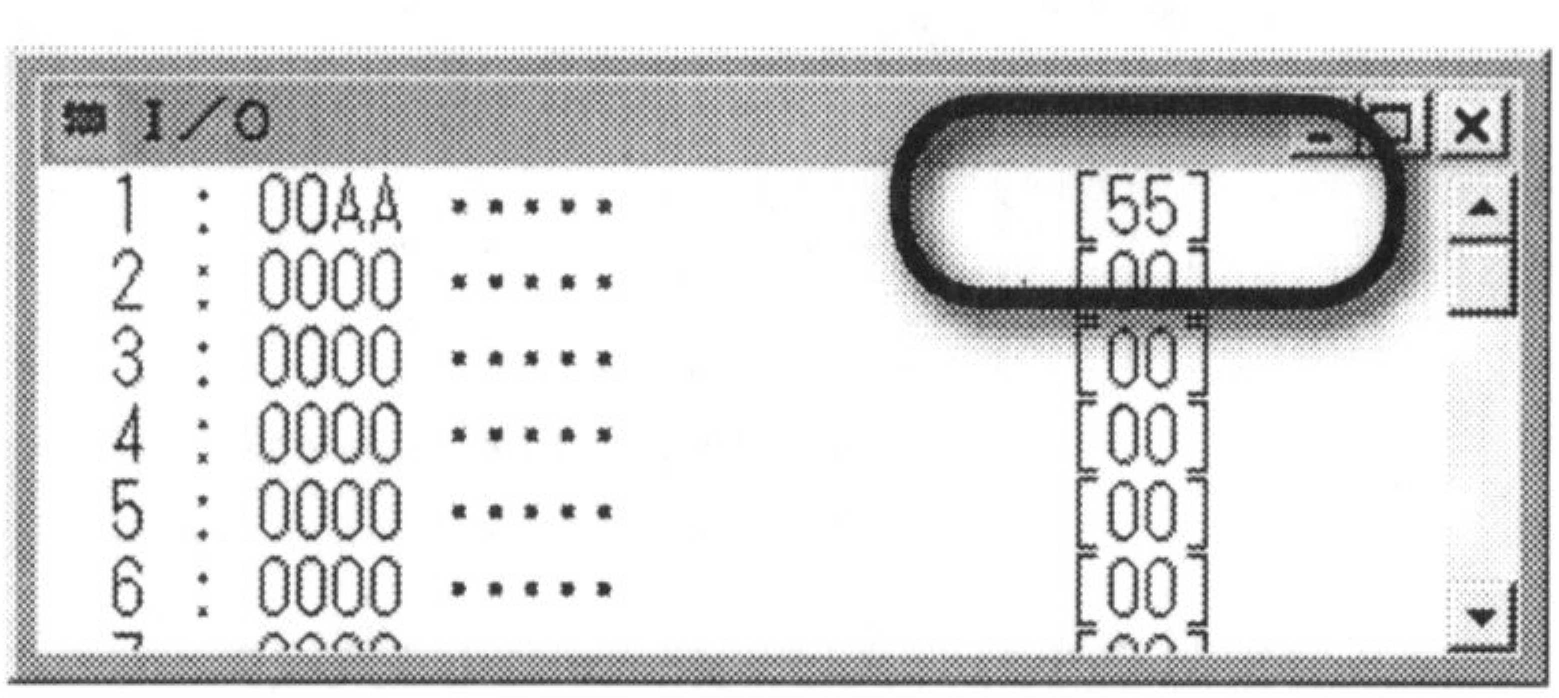


図7-19　出力プログラムの実行結果

⑤続けて、入力ポートの実験をしてみましょう。今度は、I/OウィンドウのNo.2を、I/Oアドレス25Hに設定します。出力ポートと異なる点は、あらかじめ入力データを準備しておく点です。

このときに、I/Oボタンがノンアクティブになっていて、一瞬パニックになることがあります。その時はあわて

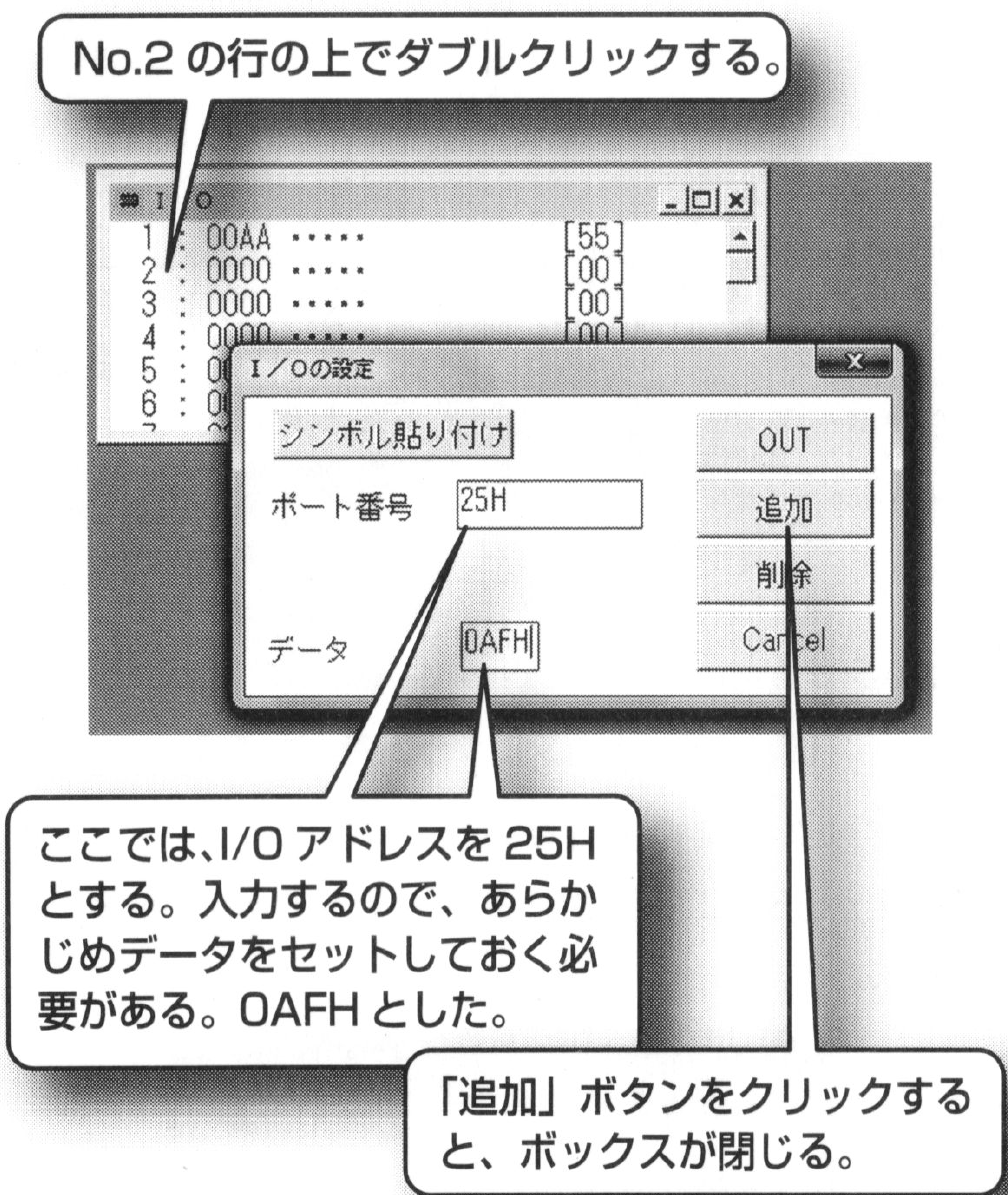


図7-20　入力ポートの設定

ずに、Z-Visionのウィンドウを最大にしてください。I/Oウィンドウは最小化されて、既に表示されています。

⑥ダンプウィンドウで、「DB 25 76」を入力します。これは、I/Oアドレス25HのI/Oポートから、1バイトのデータを取り込んで、Aレジスタに入れるプログラムです。

CPUは、まだI/Oポートの内容を知りません。

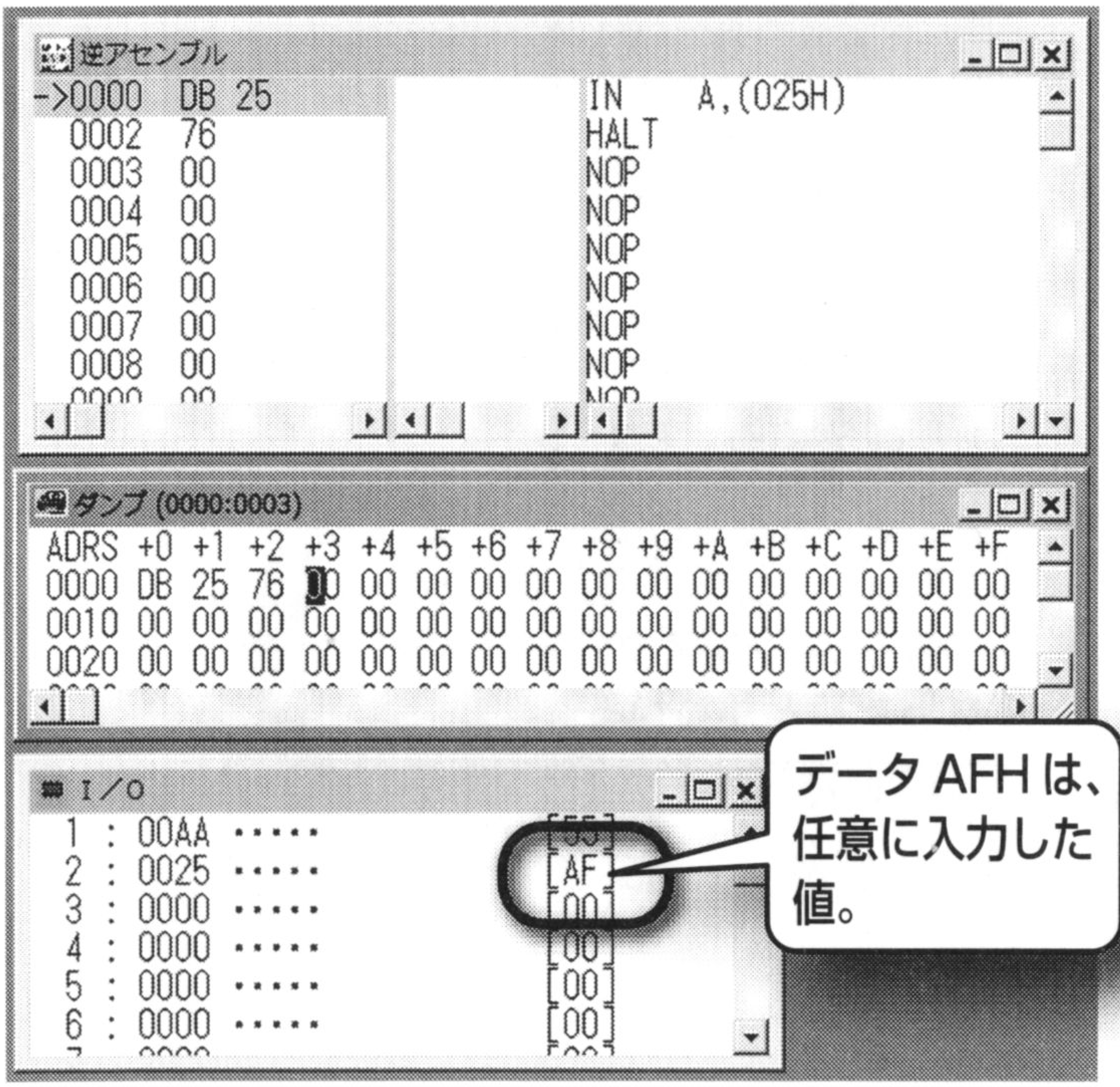


図7-21　入力プログラムを入力する

⑦わずか2行のプログラムですが、これを実行すると、レジスタウィンドウのAレジスタが、AFHに変わるのがわかります。I/Oポートから、データが入力されたという意味です。



図7-22　入力プログラムの実行

⑧なお、I/Oウィンドウについて補足しますが、I/Oアドレスやデータをリセットするには、その行をダブルクリックして「ポート番号」欄に「0」を入力します。

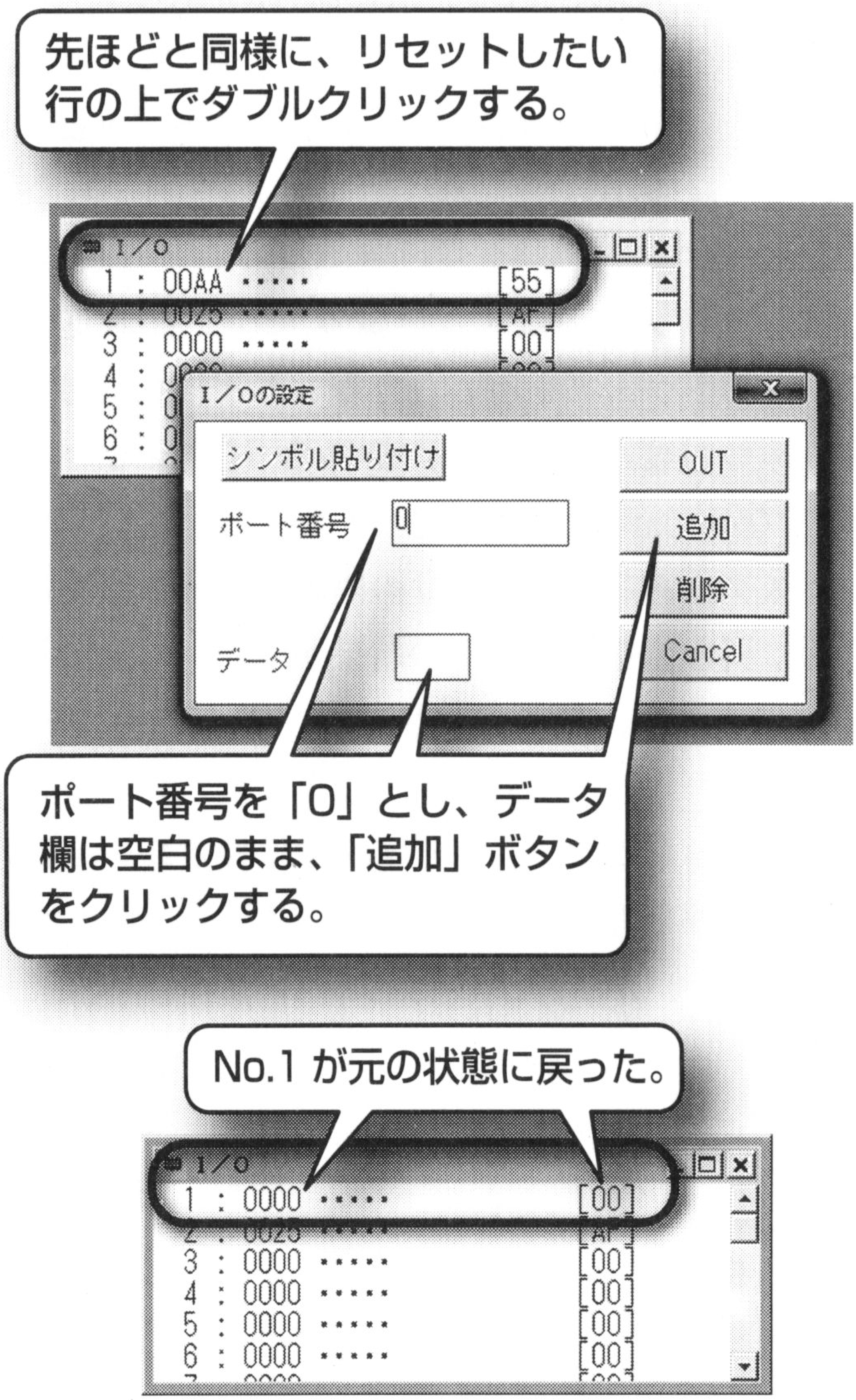


図7-23　I/Oウィンドウのデータをリセットする

7.6　8ビット算術演算命令

8ビットの算術演算は、3種類あります。

まず、Aレジスタを用いた加算と減算です。代表的なものが、ADD、ADC、SUB、SBC命令の4つです。

一方、すべてのレジスタとメモリの内容について、インクリメント（increment）とデクリメント（decrement）が可能です。

インクリメントとは1だけ加算すること（INC命令）、そしてデクリメントは1だけ減算すること（DEC命令）です。ソフトウェアを用いたカウンタとして便利です。

もう1つ重要なのがコンペア命令（CP命令）です。コンペア（compare）とは、比較すること。Aレジスタと汎用レジスタ、あるいはメモリの内容や数値と比較します。一時的に引き算を行い、その結果がフラグの変化として現れます。これを、次の条件付きジャンプ命令で利用します。

コンペア命令では、Aレジスタや比較される相手の内容は、一切影響を受けません。そのため、数値の大小関係やイコール（＝）かどうかの判定に用います。

ADD A,(HL)　86

Aレジスタの内容と、HLレジスタペアが示すアドレスの内容を加え、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

add location (HL) to accumulator

オーバーフローフラグ

ADD A,r

Aレジスタの内容と、rレジスタの内容を加え、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	87	80	81	82	83	84	85

add register r to accumulator

オーバーフローフラグ

ADD A,n　C6 n

Aレジスタの内容と1バイトの値nを加え、結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

add value n to accumulator

オーバーフローフラグ

図7-24　ADD命令

ADC A,(HL)

Aレジスタの内容に、HLレジスタペアが示すアドレスの内容とキャリーフラグの値を加え、その結果をAレジスタに入れる。

8E

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

add with carry location (HL) to accumulator

オーバーフローフラグ

ADC A,r

Aレジスタの内容に、rレジスタの内容とキャリーフラグの値を加え、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	8F	88	89	8A	8B	8C	8D

add with carry register r to accumulator

オーバーフローフラグ

ADC A,n

Aレジスタの内容に、1バイトの値nとキャリーフラグの値を加え、その結果をAレジスタに入れる。

CE
n

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

add with carry value n to accumulator

オーバーフローフラグ

図7-25　ADC命令

SUB (HL)　96

Aレジスタの内容から、HLレジスタペアが示すアドレスの内容を引き、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

subtract location (HL) from accumulator

オーバーフローフラグ

SUB r

Aレジスタの内容からrレジスタの内容を引き、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

r→	A	B	C	D	E	H	L
機械語	97	90	91	92	93	94	95

subtract register r from accumulator

オーバーフローフラグ

SUB n　D6 n

Aレジスタの内容から1バイトの値nを引き、結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

subtract value n from accumulator

オーバーフローフラグ

図7-26　SUB命令

SBC A,(HL)

Aレジスタの内容から、HLレジスタペアが示すアドレスの内容とキャリーフラグの値を引き、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

9E

subtract location (HL) from accumulator with carry

オーバーフローフラグ

SBC A,r

Aレジスタの内容から、rレジスタの内容とキャリーフラグの値を引き、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

r→	A	B	C	D	E	H	L
機械語	9F	98	99	9A	9B	9C	9D

subtract register r from accumulator with carry

オーバーフローフラグ

SBC A,n

Aレジスタの内容から、1バイトの値nとキャリーフラグの値を引き、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

DE n

subtract value n from accumulator with carry

オーバーフローフラグ

図7-27　SBC命令

increment register r

INC r

r レジスタの内容を＋1 する。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	◆

r→	A	B	C	D	E	H	L
機械語	3C	04	0C	14	1C	24	2C

オーバーフローフラグ

INC (HL)

HL レジスタペアが示すアドレスの内容を＋1 する。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	◆

34

increment location (HL)

オーバーフローフラグ

図7-28　INC命令

decrement register r

DEC r

r レジスタの内容を－1 する。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	◆

r→	A	B	C	D	E	H	L
機械語	3D	05	0D	15	1D	25	2D

オーバーフローフラグ

DEC (HL) 35

HL レジスタペアが示すアドレスの内容を－1 する。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	◆

decrement location (HL)

オーバーフローフラグ

図7-29　DEC命令

CP r

Aレジスタの内容からrレジスタの内容を引き、その結果をAレジスタに入れる。Aレジスタの内容は変化しない。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

r→	A	B	C	D	E	H	L
機械語	BF	B8	B9	BA	BB	BC	BD

compare register r with accumulator

オーバーフローフラグ

CP (HL)　　BE

Aレジスタの内容から、HLレジスタペアが示すアドレスの内容を引き、その結果をAレジスタに入れる。Aレジスタの内容は変化しない。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

compare location (HL) with accumulator

オーバーフローフラグ

CP n　　FE n

Aレジスタの内容から1バイトの値nを引き、その結果をAレジスタに入れる。Aレジスタの内容は変化しない。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

compare value n with accumulator

オーバーフローフラグ

図7-30　CP命令

ここで、ADD命令とSUB命令を使ってみましょう。

機械語は「3E 80 06 80 80 D6 80 D6 80 76」、そして図7-31 ～図7-36が動作の様子です。

このプログラムの動作は、まずAレジスタとBレジスタに80H、つまり10進数で128（または128D）をロードします。これらを加算すると256ですから、8ビットをはみ出してしまいます。これがオーバーフローです。この時点で、Aレジスタの値は、見かけ上「0」です。

その後、Aレジスタから、2回に分けて80H（128D）ずつ引き算します。1回目の引き算で、仮想の上の桁から1ビット借りてきますので、値は256－128＝128(80H)に戻ります。この「借りる」ことをボローと言います。さらに80Hを引くと、Aレジスタの値は0になります。

この実験では、フラグの動きが活発です。

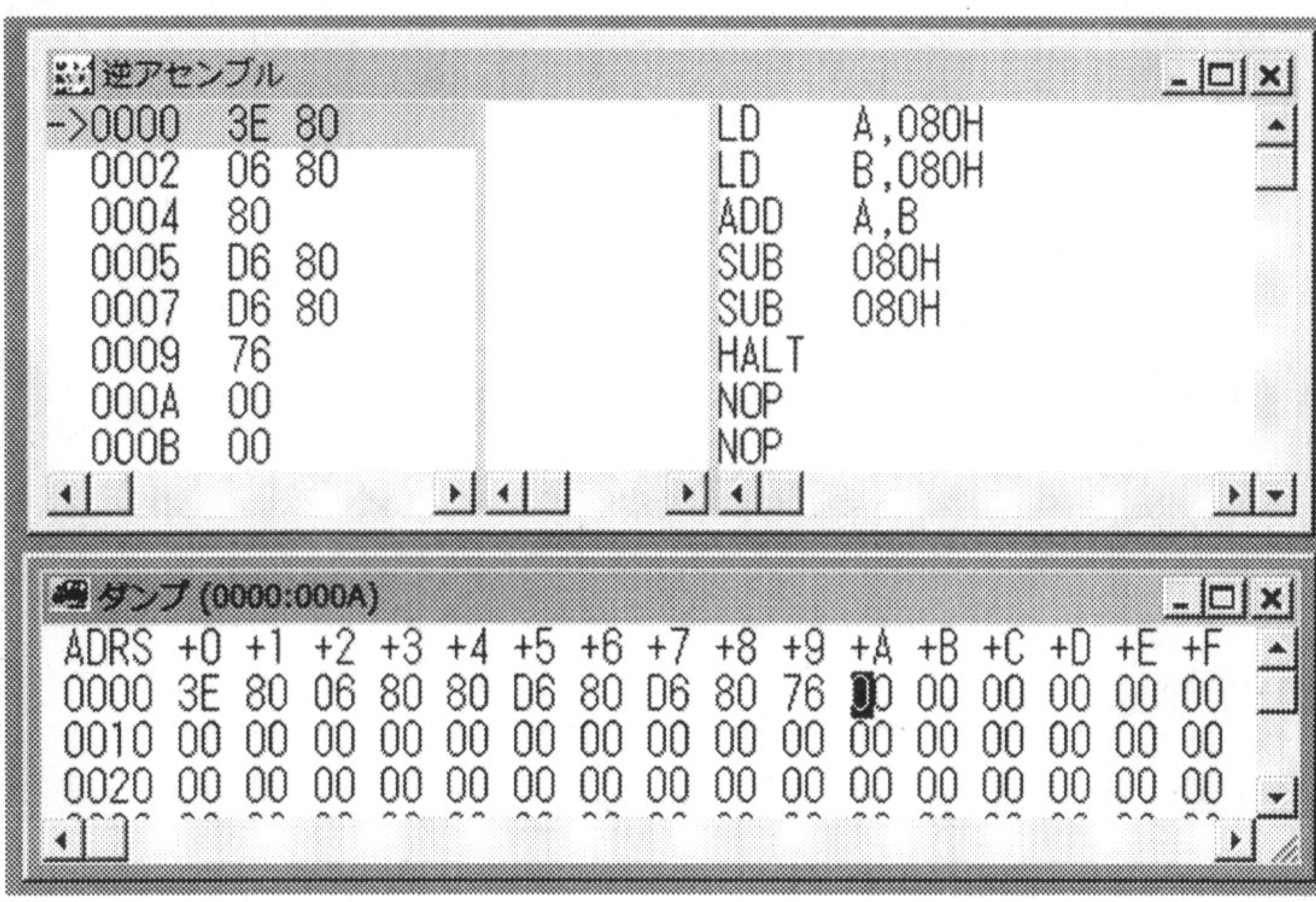


図7-31　ADD/SUB命令を使う

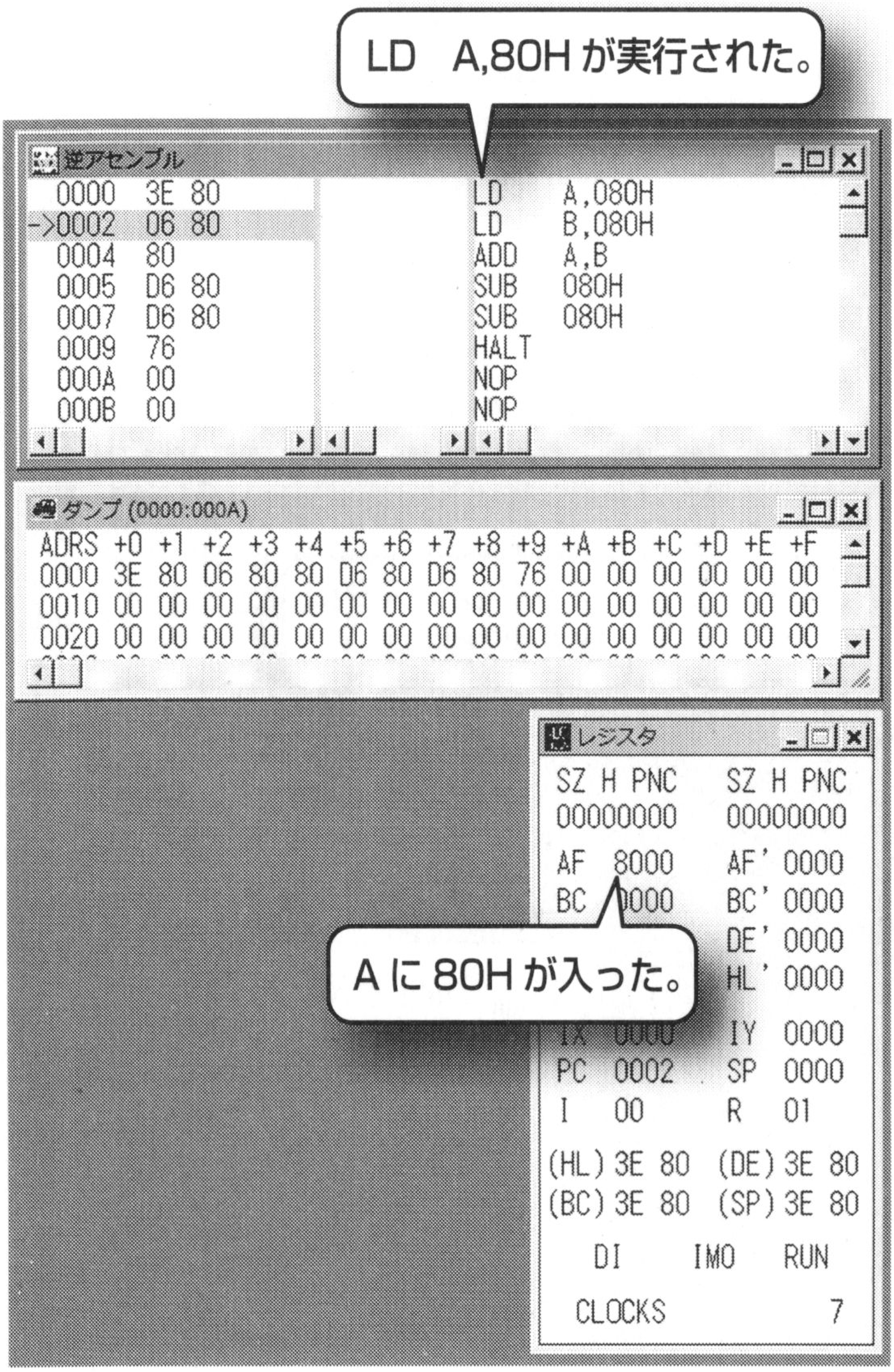


図7-32　1行目を実行する

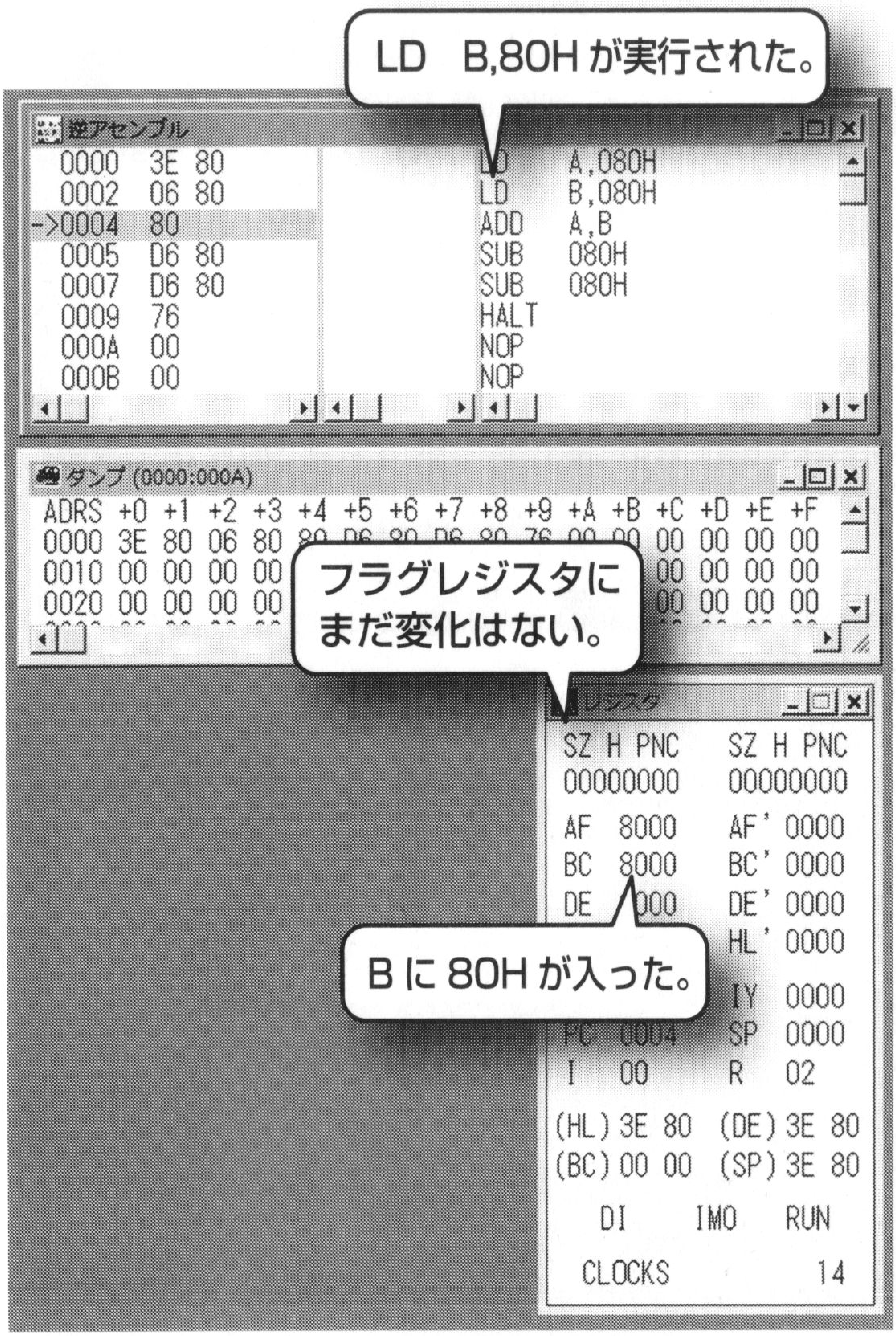


図 7-33　2行目を実行する

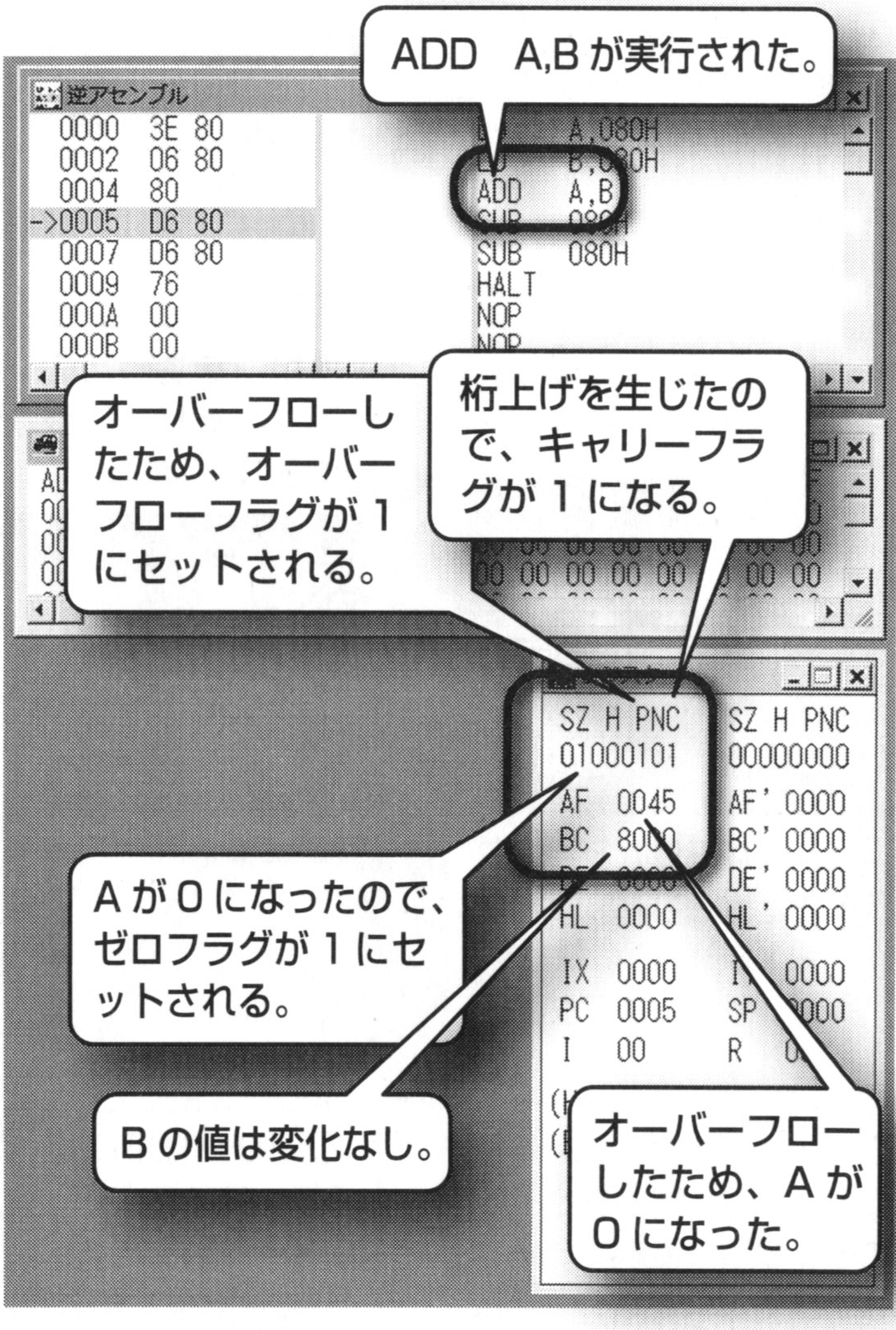


図7-34　3行目を実行する

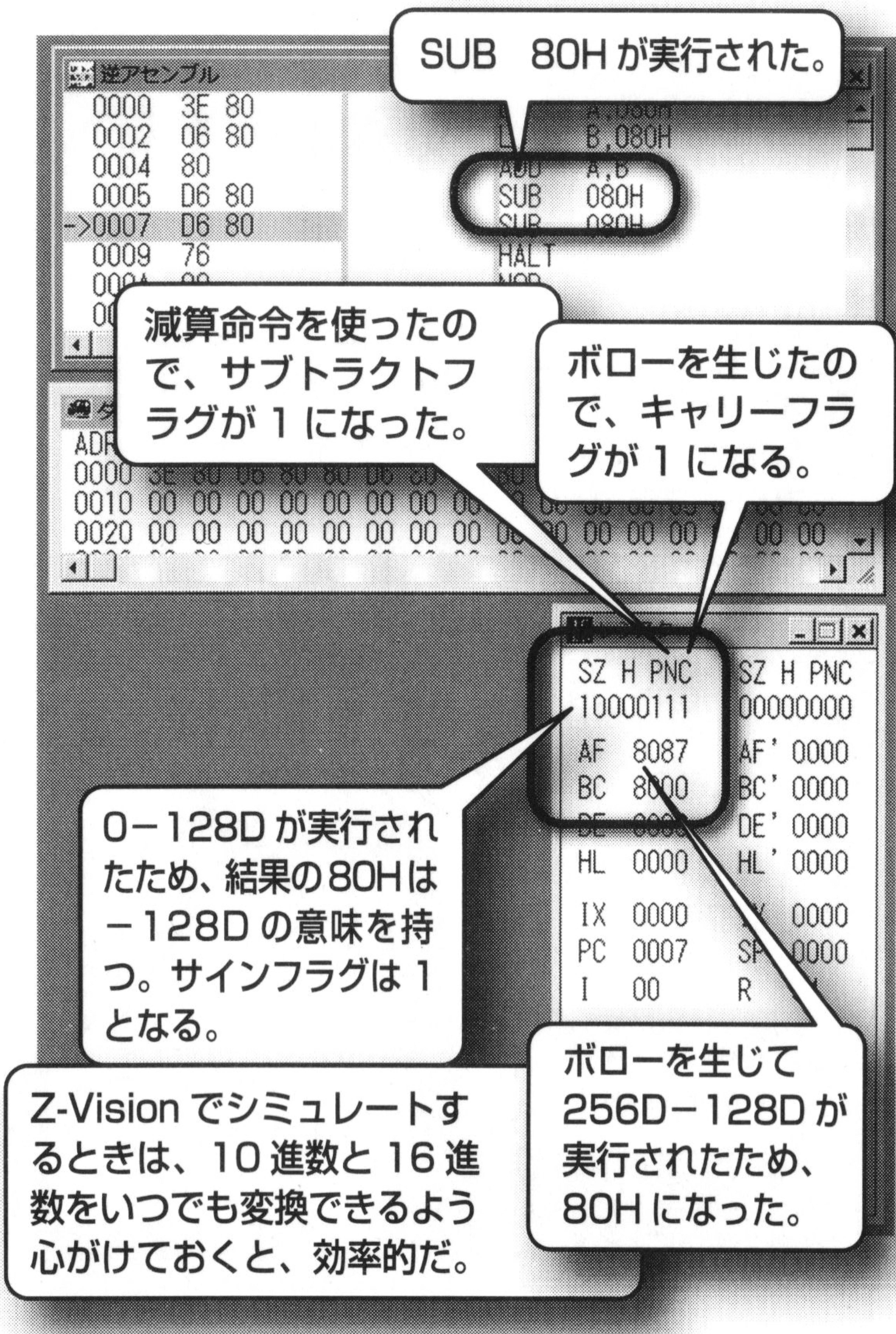


図7-35　4行目を実行する

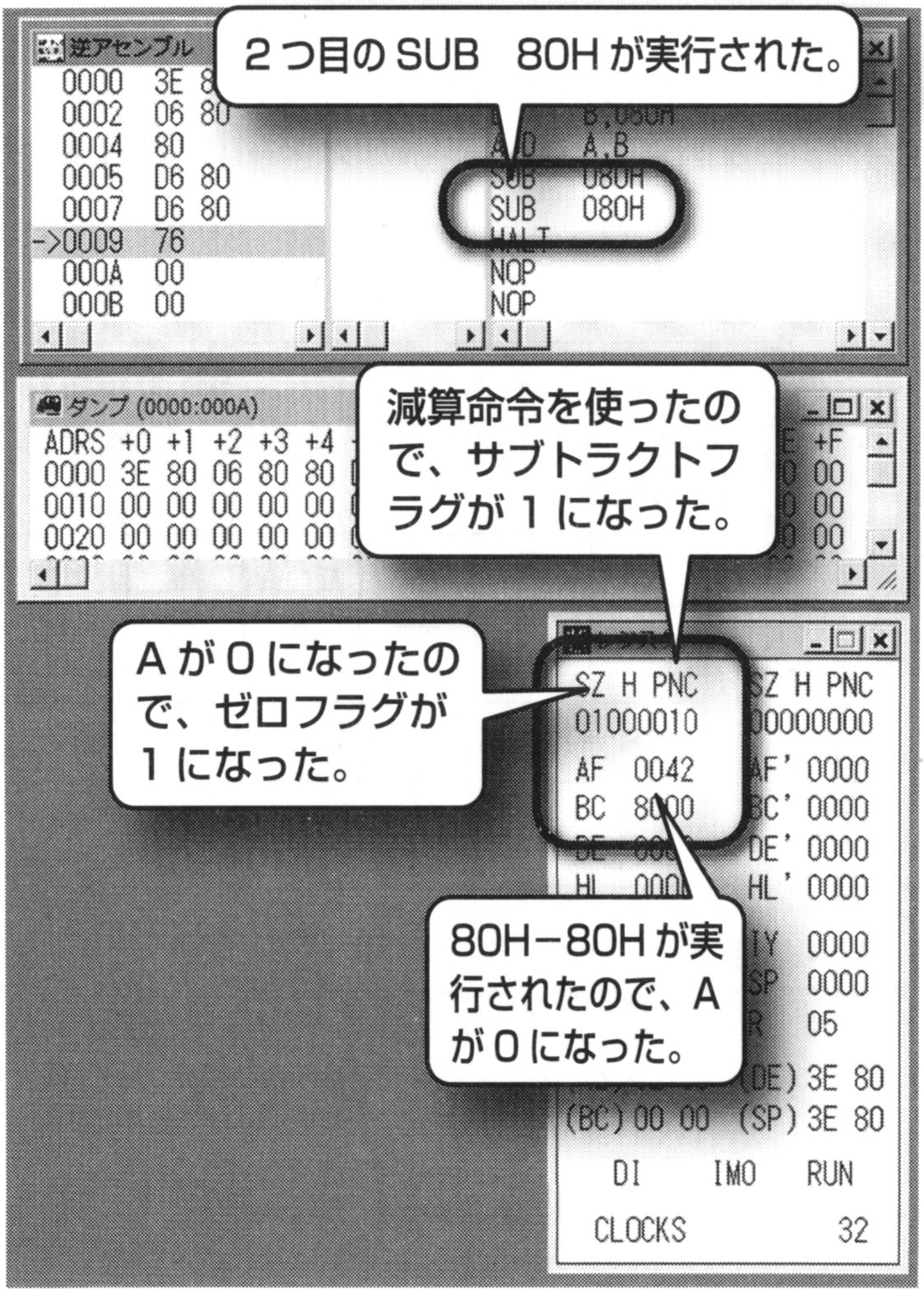


図7-36　5行目を実行する

7.7　16ビット算術演算命令

算術演算の多くは8ビットですが、16ビットレジスタペア同士に限って、算術演算ができます。

命令の種類は少なく、加算についてはADD命令とADC命令、減算についてはSBC命令だけです。

なお、フラグの変化のしかたが、8ビット算術演算命令と大きく異なりますので注意しましょう。

ADD HL,ss

HLレジスタペアの内容に、ssで指定されるレジスタペアの内容を加え、結果をHLに入れる。

S	Z	H	P/V	N	CY
◆	◆	↑↓	◆	0	↑↓

ss→	BC	DE	HL	SP
機械語	09	19	29	39

add register pair ss to HL

図7-37　16ビットADD命令

ADC HL,ss

HL レジスタペアの内容に、ss で指定されるレジスタペアの内容とキャリーフラグの値を加え、結果を HL に入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	0	↑↓

ss→	BC	DE	HL	SP
機械語	ED 4A	ED 5A	ED 6A	ED 7A

add with carry register pair ss to HL

オーバーフローフラグ

SBC HL,ss

HL レジスタペアの内容から、ss で指定されるレジスタペアの内容とキャリーの値を引き、結果を HL に入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	1	↑↓

ss→	BC	DE	HL	SP
機械語	ED 42	ED 52	ED 62	ED 72

subtract register pair ss from HL with carry

オーバーフローフラグ

図7-38　16ビットADC/SBC命令

INC ss

ss で指定されるレジスタペアの内容を＋1 する。　◆フラグは変化しない。

ss→	BC	DE	HL	SP
機械語	03	13	23	33

increment register pair ss

DEC ss

ss で指定されるレジスタペアの内容を－1 する。　◆フラグは変化しない。

ss→	BC	DE	HL	SP
機械語	0B	1B	2B	3B

decrement register pair ss

図7-39　16ビットINC/DEC命令

7.8　8ビット論理演算命令

論理演算はAレジスタとの間だけで行われ、論理積（AND命令）、論理和（OR命令）、排他論理和（XOR命令）の3種類があります。

図7-40は、AレジスタとBレジスタの演算結果を、ビット単位で一覧にしたものです。

(a) AND

演算前A	B	演算後A
1	1	1
1	0	0
0	1	0
0	0	0

(b) OR

演算前A	B	演算後A
1	1	1
1	0	1
0	1	1
0	0	0

(c) XOR

演算前A	B	演算後A
1	1	0
1	0	1
0	1	1
0	0	0

図7-40　ANDとORとXORの意味

AND r

Aレジスタの内容とrレジスタの内容を論理積演算し、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	1	↑↓	0	0

r→	A	B	C	D	E	H	L
機械語	A7	A0	A1	A2	A3	A4	A5

logical AND of register r and accumulator

パリティフラグ

AND (HL) — A6

Aレジスタの内容とHLレジスタペアが示すアドレスの内容を論理積演算し、結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	1	↑↓	0	0

logical AND of location (HL) and accumulator

パリティフラグ

AND n — E6

Aレジスタの内容と1バイトの値nを論理積演算し、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	1	↑↓	0	0

logical AND of value n and accumulator

パリティフラグ

図7-41　AND命令

OR r

Aレジスタの内容とrレジスタの内容を論理和演算し、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	0

r→	A	B	C	D	E	H	L
機械語	B7	B0	B1	B2	B3	B4	B5

logical OR of register r and accumulator

パリティフラグ

OR (HL)　B6

Aレジスタの内容と、HLレジスタペアが示すアドレスの内容を論理和演算して、結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	0

logical OR of location (HL) and accumulator

パリティフラグ

OR n　F6 n

Aレジスタの内容と1バイトの値nを論理和演算し、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	0

logical OR of value n and accumulator

パリティフラグ

図7-42　OR命令

XOR r

Aレジスタの内容とrレジスタの内容を排他論理和演算し、結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	0

r→	A	B	C	D	E	H	L
機械語	AF	A8	A9	AA	AB	AC	AD

logical exclusive OR of register r and accumulator

パリティフラグ

XOR (HL)

Aレジスタの内容と、HLレジスタペアが示すアドレスの内容を排他論理和演算して、結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	0

AE

logical exclusive OR of location (HL) and accumulator

パリティフラグ

XOR n

Aレジスタの内容と1バイトの値nを排他論理和演算し、その結果をAレジスタに入れる。

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	0

EE nn

logical exclusive OR of value n and accumulator

パリティフラグ

図7-43　XOR命令

7.9　ジャンプ命令

ここで取り上げるジャンプ命令と、7.10節に述べるコール/リターン命令は、ユーザがプログラムの中のさまざまな場所（アドレス）へ移動をするために使います。

ジャンプ命令には、無条件で指定したアドレスへジャンプするものと、直前の命令実行でフラグがセットされた場合に、そのフラグの内容によってジャンプするかどうかが決まるものがあります。

図7-46～図7-52では、無条件ジャンプ命令と条件付きジャンプ命令を同時に使い、永久ループのプログラムを走らせます。

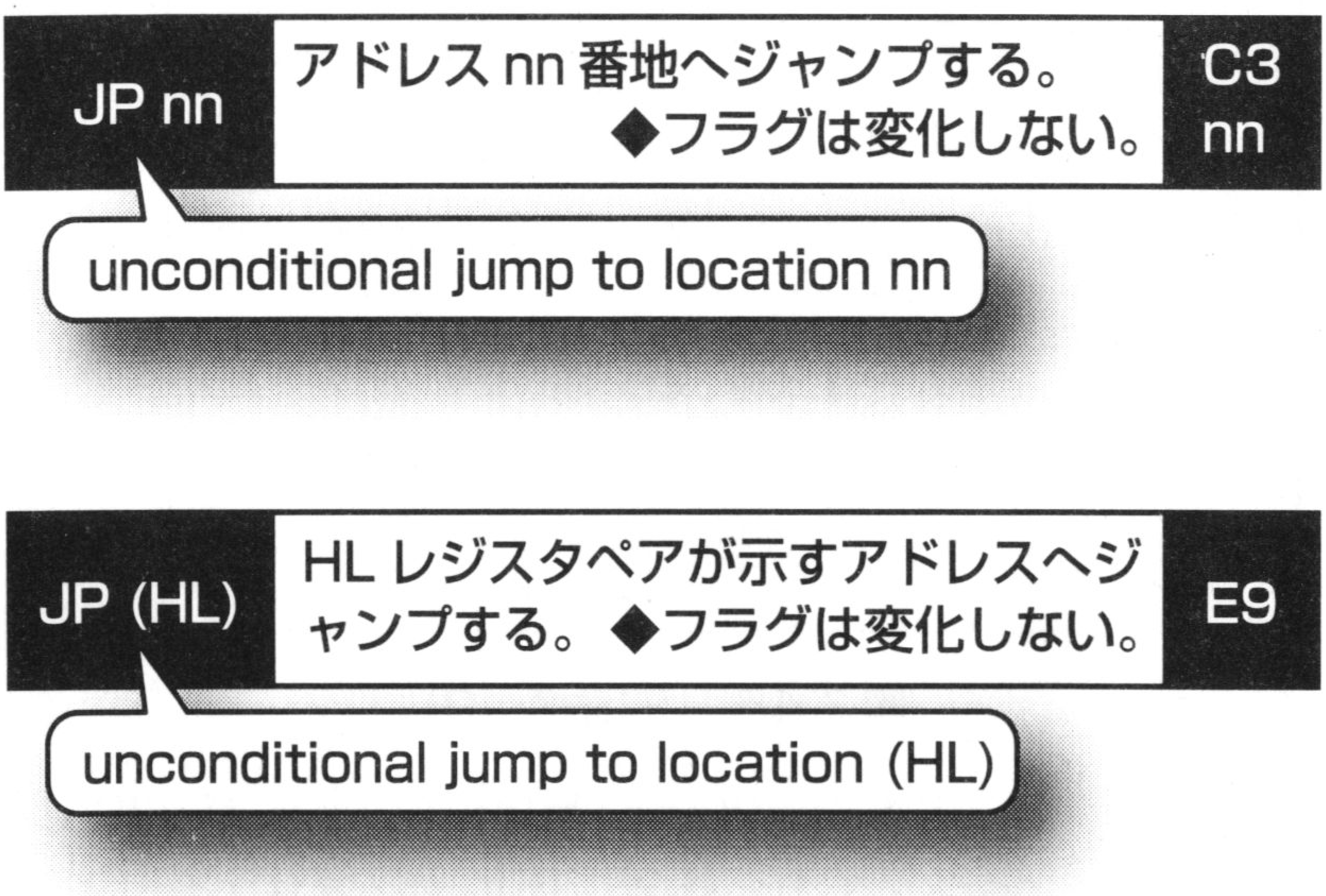


図7-44　JP命令（1）

jump to location nn if condition cc is true

JP cc,nn

判定条件 cc が「真」、つまり YES であるとき、アドレス nn へジャンプする。真でない場合は、次の命令に進む。

◆フラグは変化しない。

cc	機械語	判定条件の意味
NZ	C2 nn	演算結果が 0 でないなら
Z	CA nn	演算結果が 0 なら
NC	D2 nn	キャリーフラグが 0 なら
C	DA nn	キャリーフラグが 1 なら
PO	E2 nn	パリティフラグが奇数なら
PE	EA nn	パリティフラグが偶数なら
P	F2 nn	演算結果がプラスなら
M	FA nn	演算結果がマイナスなら

図7-45　JP命令（2）

ダンプウィンドウで、0000H番地から、「3E 02 3D CA 00 00 C3 02」と入力します。このプログラムは、初期値２が与えられたＡレジスタが０になるまでデクリメント（１だけ減算）され、０になった時点で再び初期値２が与えられるというものです。

そして、ここでの判定対象はゼロフラグです。

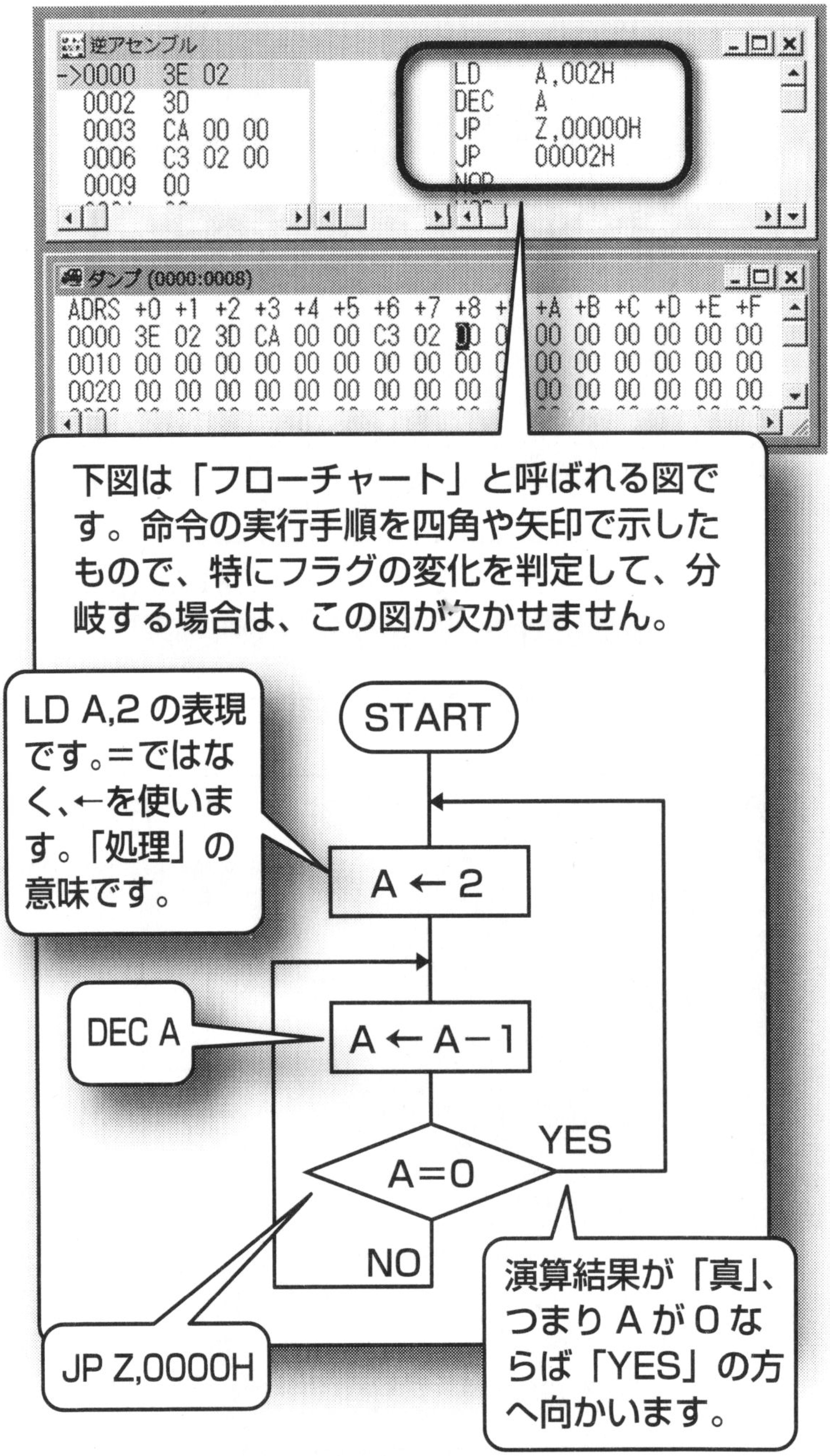


図7-46　JP命令とフローチャート



図7-47　1行目を実行する

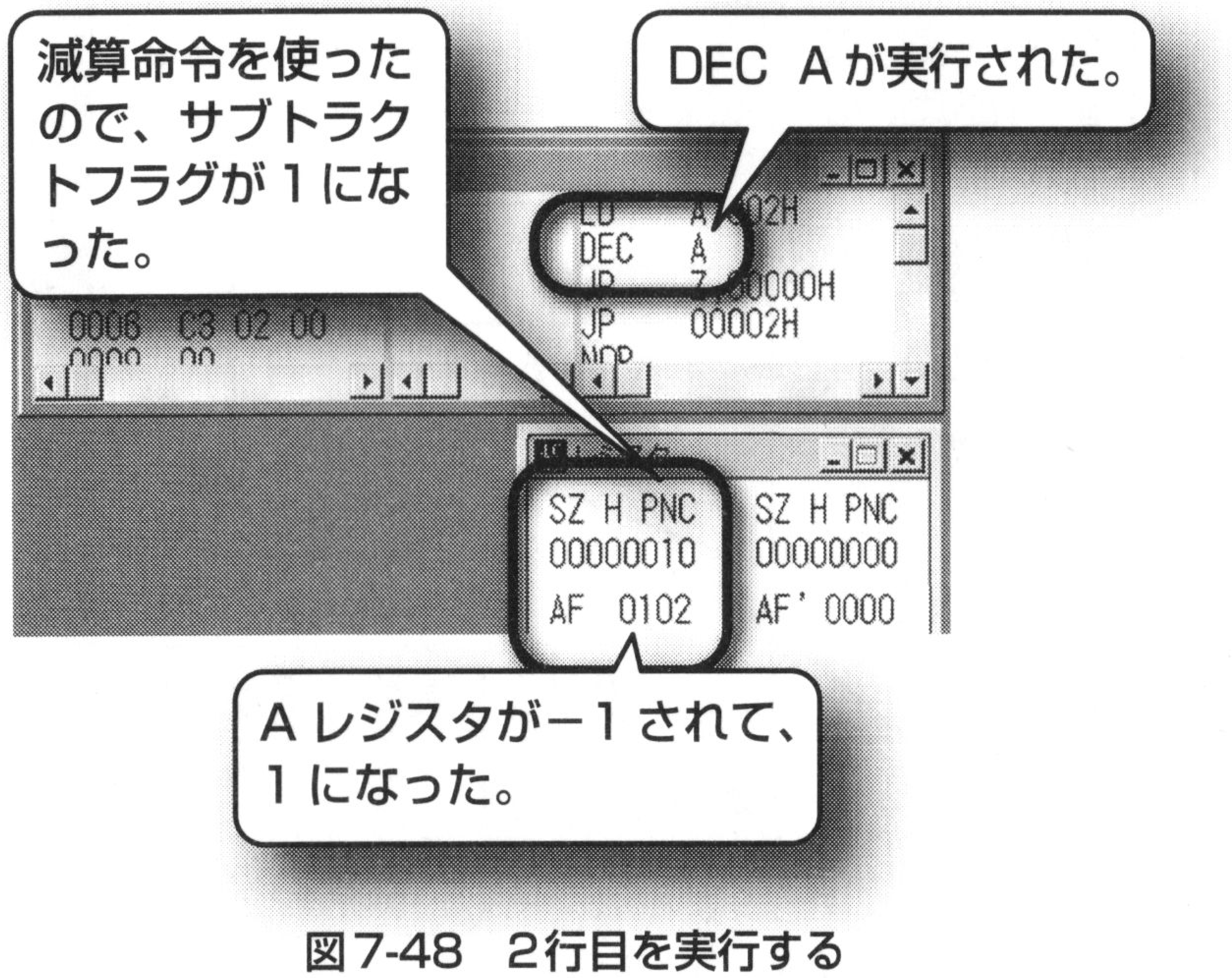


図7-48　2行目を実行する

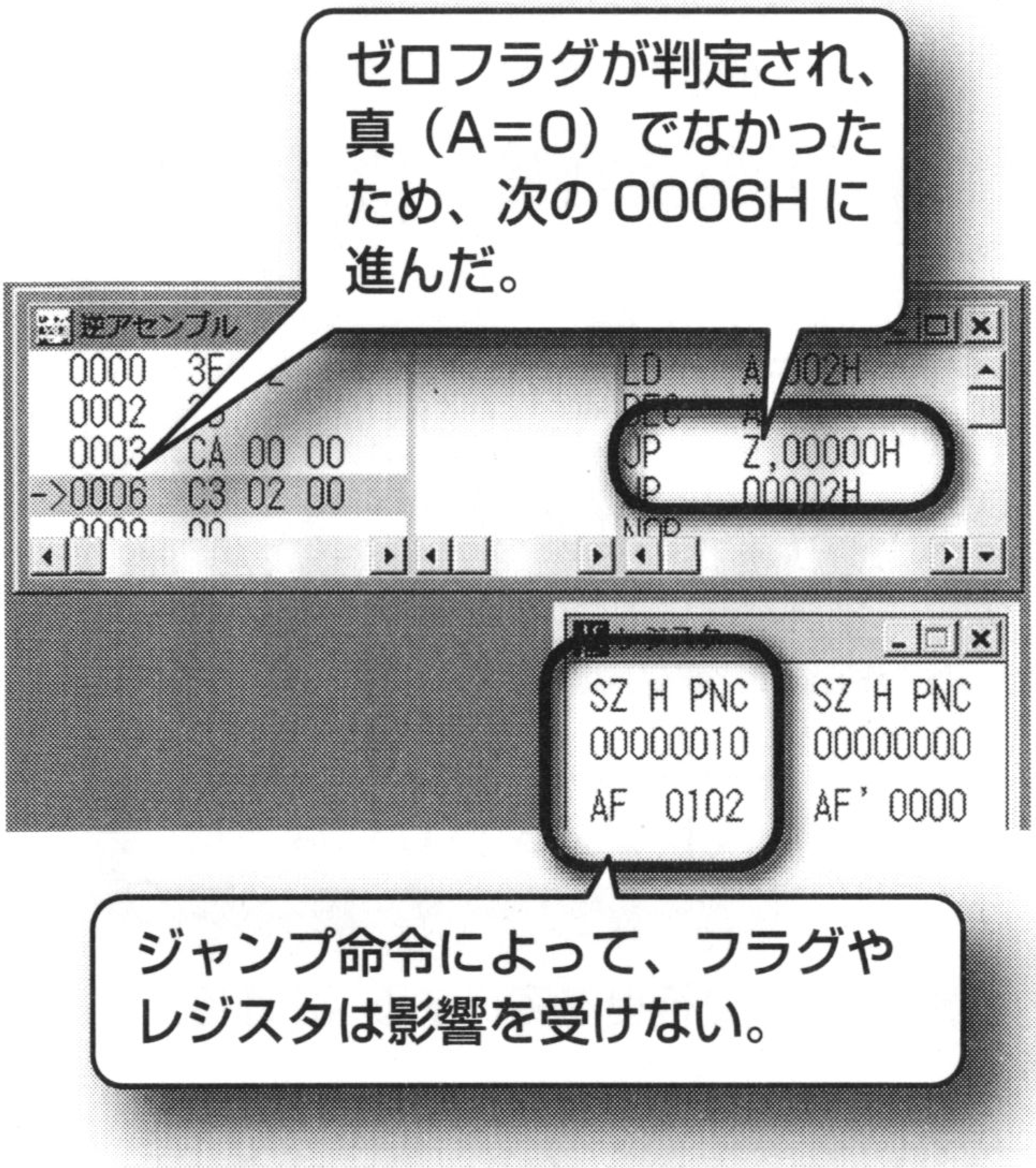


図7-49　3行目を実行する

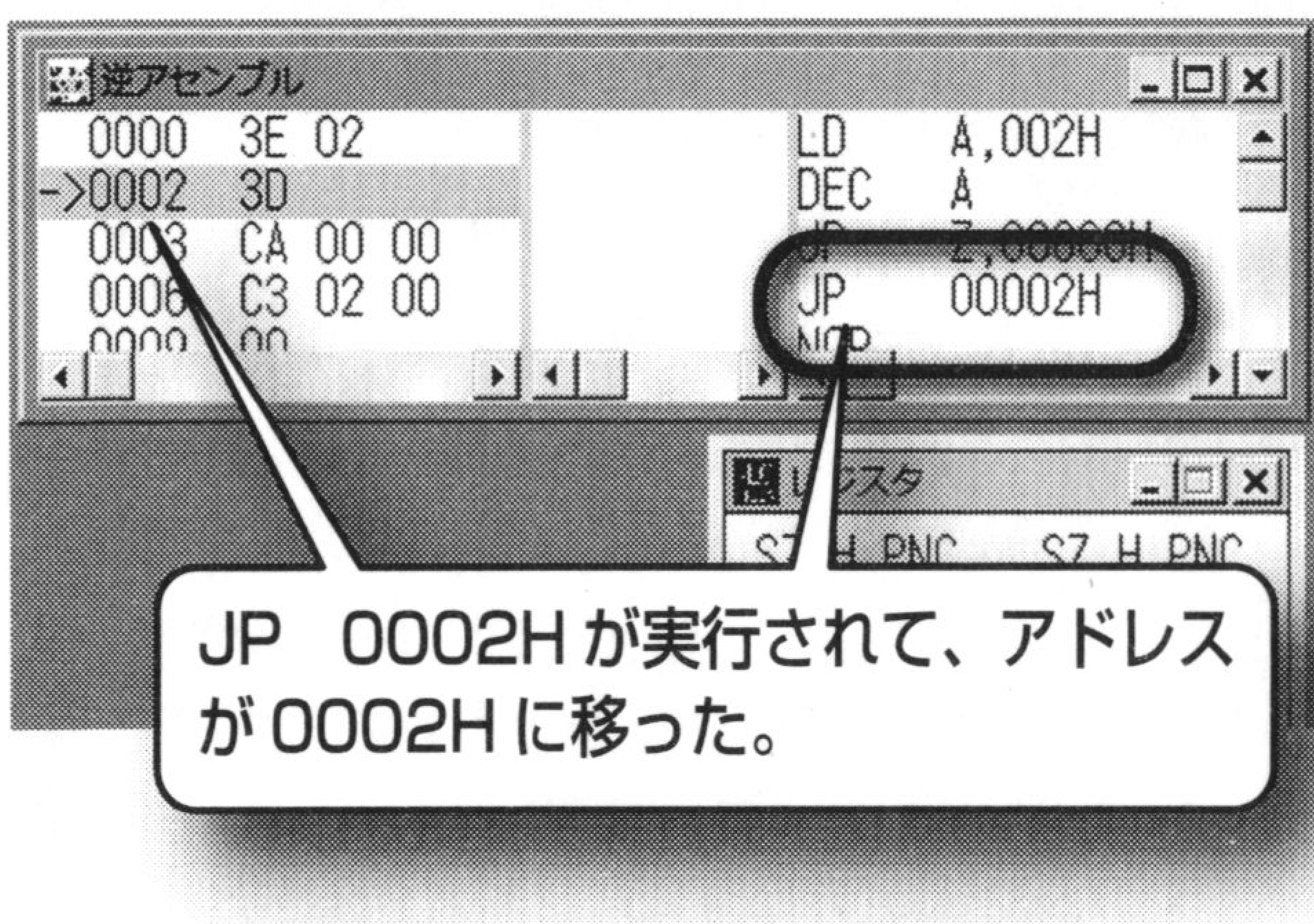


図7-50　4行目を実行する

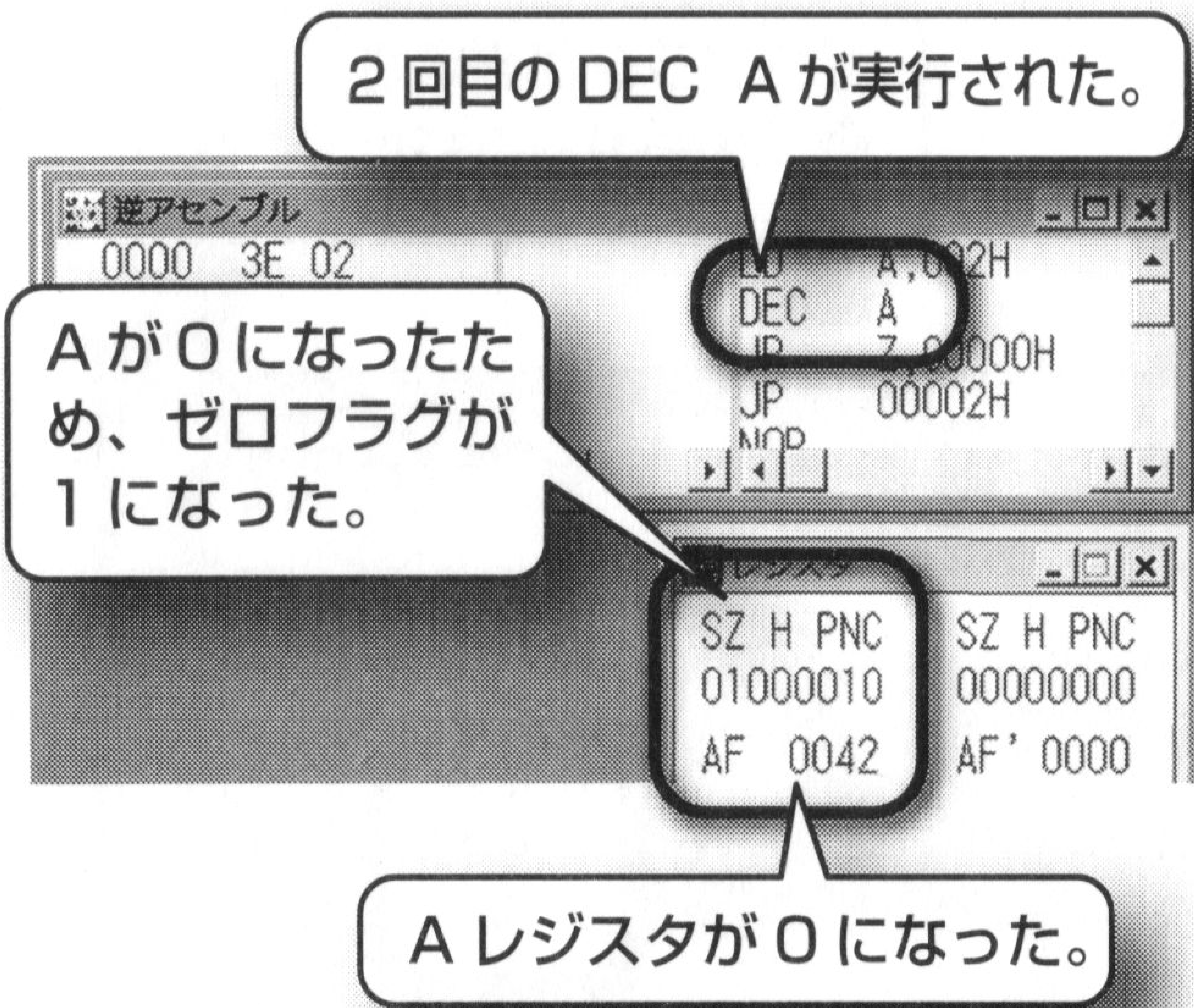


図7-51　2行目を実行する（2回目）

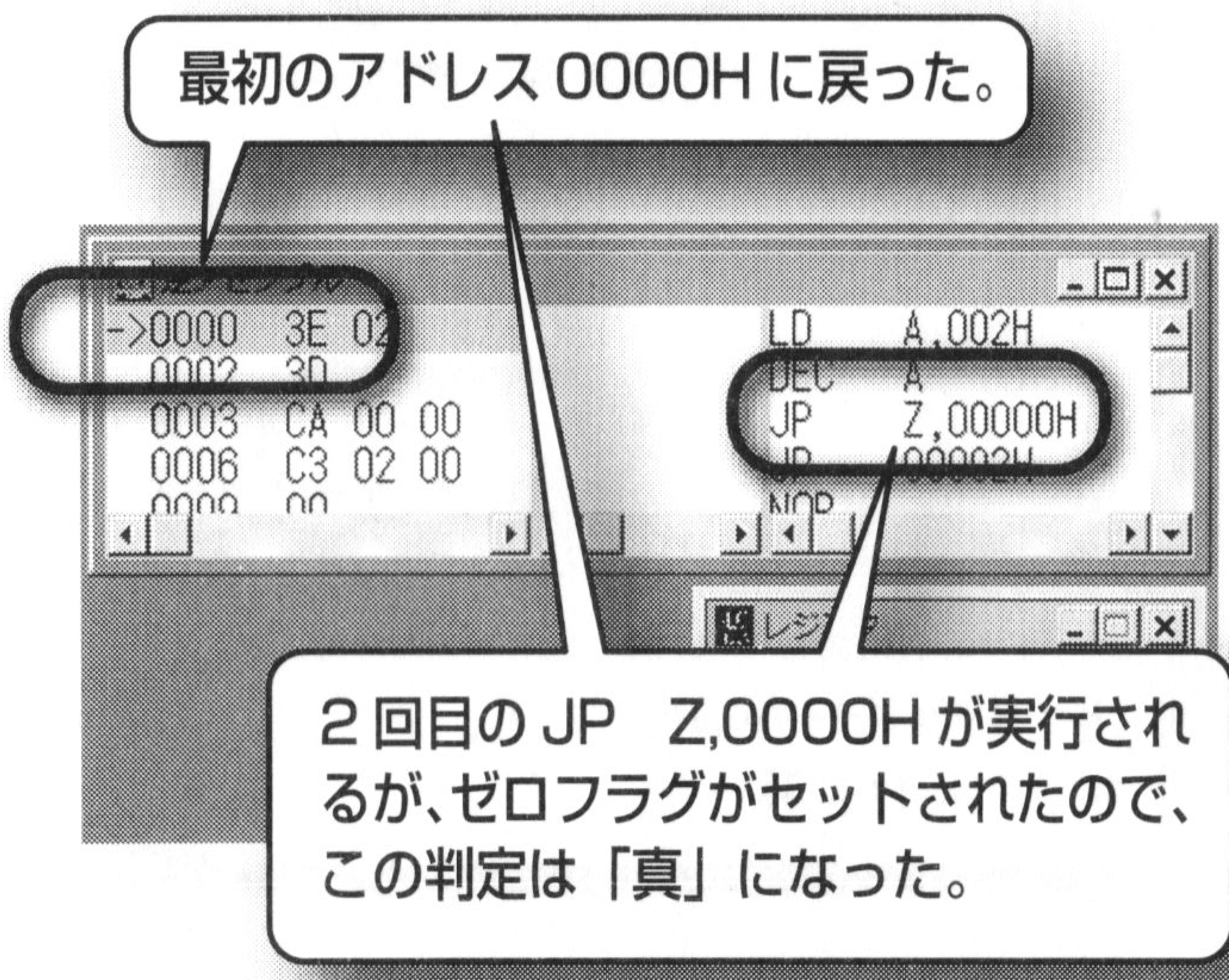
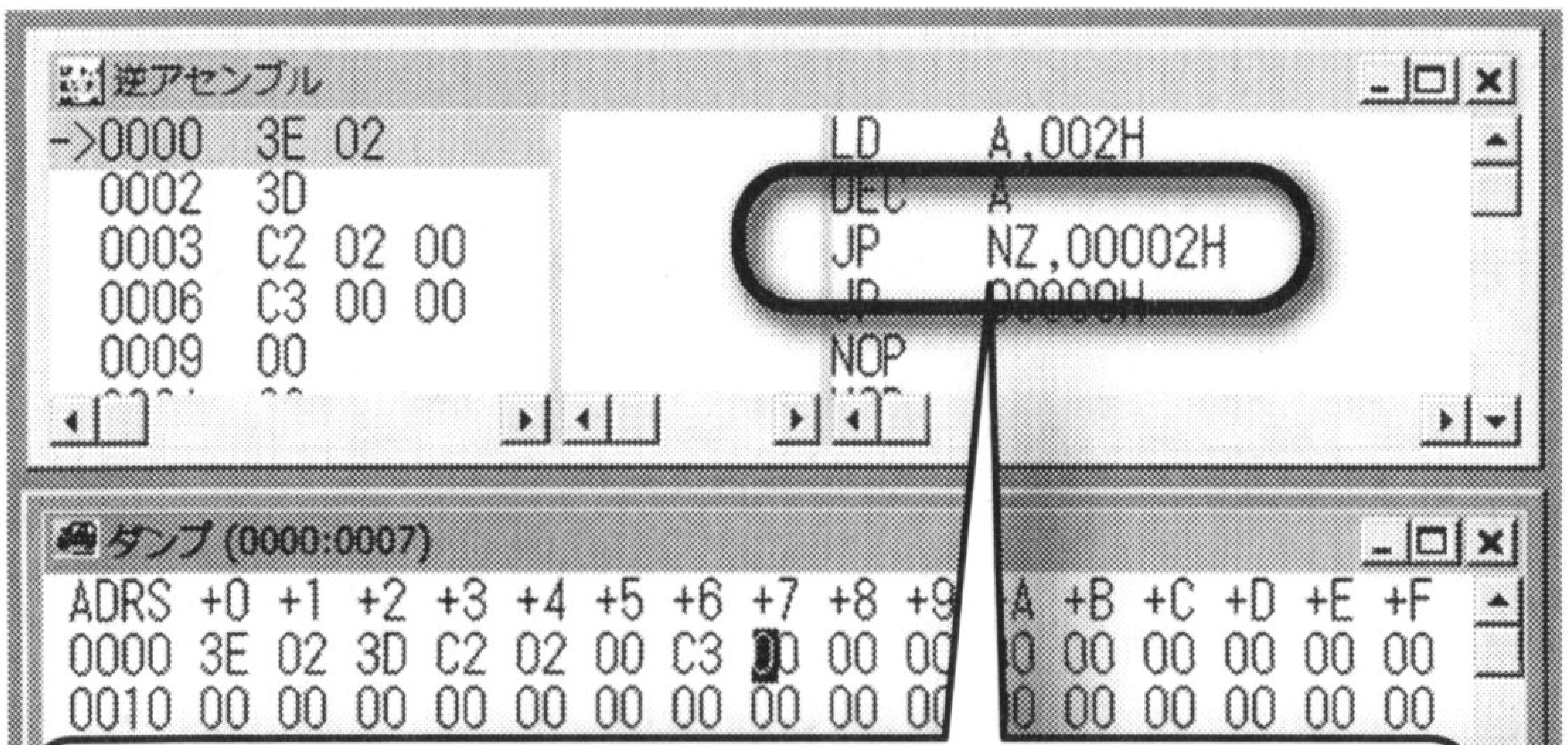


図7-52　3行目を実行する（2回目）



下図は、JP　Z,nn 命令を JP　NZ,nn 命令に置き換えたフローチャートです。動作は図7-46とまったく同じですが、判定条件と矢印の行き先が異なります。

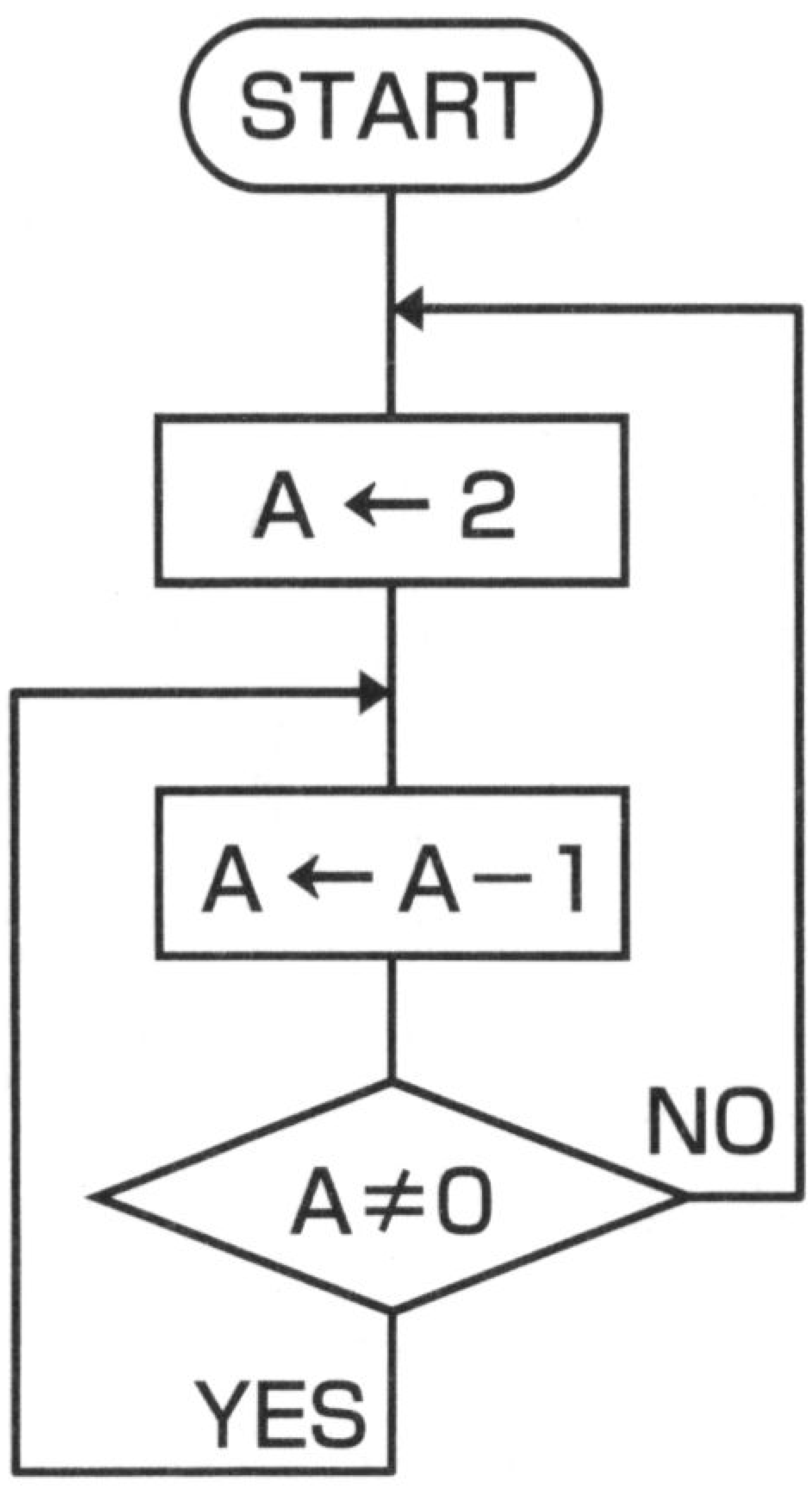


図7-53　JP　NZ,nn 命令を使う

7.10　コール/リターン命令

コール命令は、メインルーチンから離れたアドレスに配置されたサブルーチン（例えば、繰り返し利用される独立したプログラム）を走らせるための命令です。

動作としては、まずコール命令の次に配置されている命令のアドレス値が、ジャンプする直前にスタック（stack）と呼ばれるメモリ上にプッシュ（push）されます。スタックには「積み重ねる」という意味があります。あとで戻って実行する命令のアドレスをRAM上に積み重ねる、言い換えると「一時的に預ける」ことなのです。

スタックの位置は、基本的にはプログラムの冒頭にロード命令で指定します。たとえば、CALL SP,5000Hと記述すると、5000Hから－1した4FFFHから、4FFEH、4FFDH、……とさかのぼりながら2バイトずつアドレス値を記憶していきます。

つまり、スタックというメモリ領域は、通常は使用しないRAM領域の最後の方から逆向きに使われるのです。これは、よく考えられた効率的な仕組みです。また、どこまでさかのぼったのかを記憶するカウンタがスタックポインタ（SP）です。

一方、リターン命令はコール命令の逆の動作で、スタックのトップ（最も先頭）にあるデータが、ジャンプする直前にプログラムカウンタ（PC）にポップ（pop）されます。「プッシュ」はスタックにアドレス値を送ること、一方、その値を戻すのが「ポップ」です。これらの動きを、図7-56から図7-65で観察してみましょう。

命令	説明	機械語
CALL nn	無条件にアドレス nn 番地へジャンプする。このとき SP の値は、2 だけ引き算される。 ◆フラグは変化しない。	CD nn

unconditional call to subroutine at location nn

call to subroutine at location nn if condition cc is true

CALL cc,nn

判定条件 cc が「真」、つまり YES であるとき、アドレス nn へジャンプし、SP の値は 2 だけ引き算される。また、PC の上位 8 ビット値を SP − 1 で示されるメモリへ記憶、下位 8 ビット値を SP − 2 へ記憶する。一般に、RET 命令とペアで用いる。判定条件が真でない場合は、CALL の次の命令に進む。
◆フラグは変化しない。

cc	機械語	判定条件の意味
NZ	C2 nn	演算結果が 0 でないなら
Z	CA nn	演算結果が 0 なら
NC	D2 nn	キャリーフラグが 0 なら
C	DA nn	キャリーフラグが 1 なら
PO	E2 nn	パリティフラグが奇数なら
PE	EA nn	パリティフラグが偶数なら
P	F2 nn	演算結果がプラスなら
M	FA nn	演算結果がマイナスなら

図7-54　CALL命令

RET	無条件にアドレス nn 番地へジャンプする。このとき SP の値は、2 だけ引き算される。 ◆フラグは変化しない。	C9

return from subroutine

return from subroutine if condition cc is true

RET cc	判定条件 cc が「真」、つまり YES であるとき、SP の値が 2 だけ加算される。また、SP が示すアドレスの内容が PC の下位 8 ビットに戻され、SP+1 の内容が上位 8 ビットに戻される。一般に、CALL 命令とペアで用いられる。なお、判定条件が真でない場合は、RET の次の命令に進む。 ◆フラグは変化しない。

cc	機械語	判定条件の意味
NZ	C2 nn	演算結果が 0 でないなら
Z	CA nn	演算結果が 0 なら
NC	D2 nn	キャリーフラグが 0 なら
C	DA nn	キャリーフラグが 1 なら
PO	E2 nn	パリティフラグが奇数なら
PE	EA nn	パリティフラグが偶数なら
P	F2 nn	演算結果がプラスなら
M	FA nn	演算結果がマイナスなら

図7-55　RET命令

①サブルーチンの実験では、0000Hからスタートするメインルーチンとサブルーチン、そしてスタックの内容を同時に見えるようにすると、観察に便利です。

ここでは、サブルーチンを0010Hから、スタックポインタの値を5000Hとしました。サブルーチンの仕事は、I/Oポートに34Hというデータを出力するだけです。

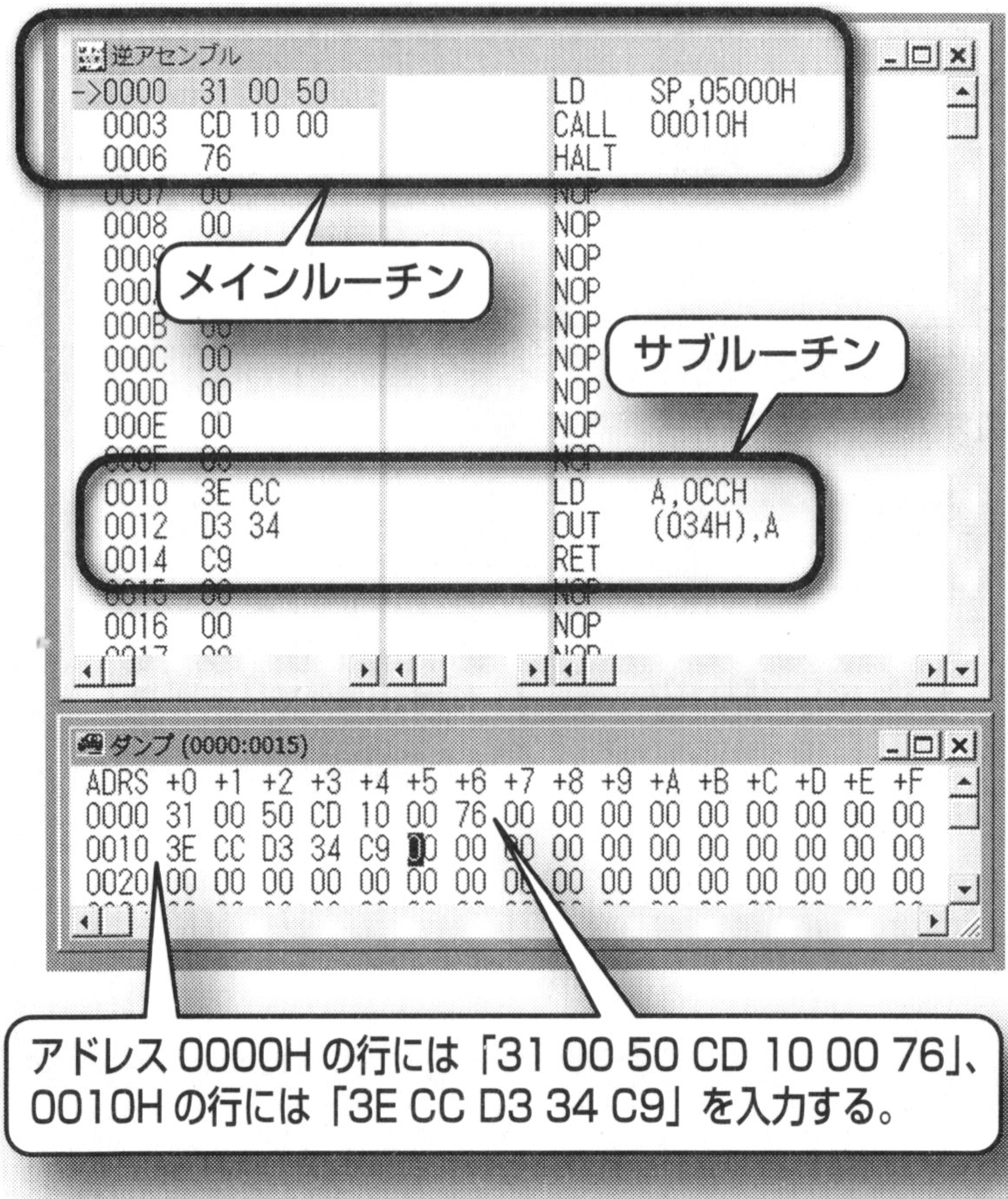


図7-56 「メイン」と「サブ」ルーチンの入力

②ところで、この場合のスタックの内容は、どのようにすれば見えるでしょうか？

Z-Visionでは、逆アセンブルウィンドウとダンプウィンドウを、同時にそれぞれ4つずつ表示することができますが、プログラムの実行中に表示内容が動くことがあって、落ち着いて観察することができません。

それよりも、特定のアドレスの値を常に表示できるウォッチウィンドウの方が便利です。

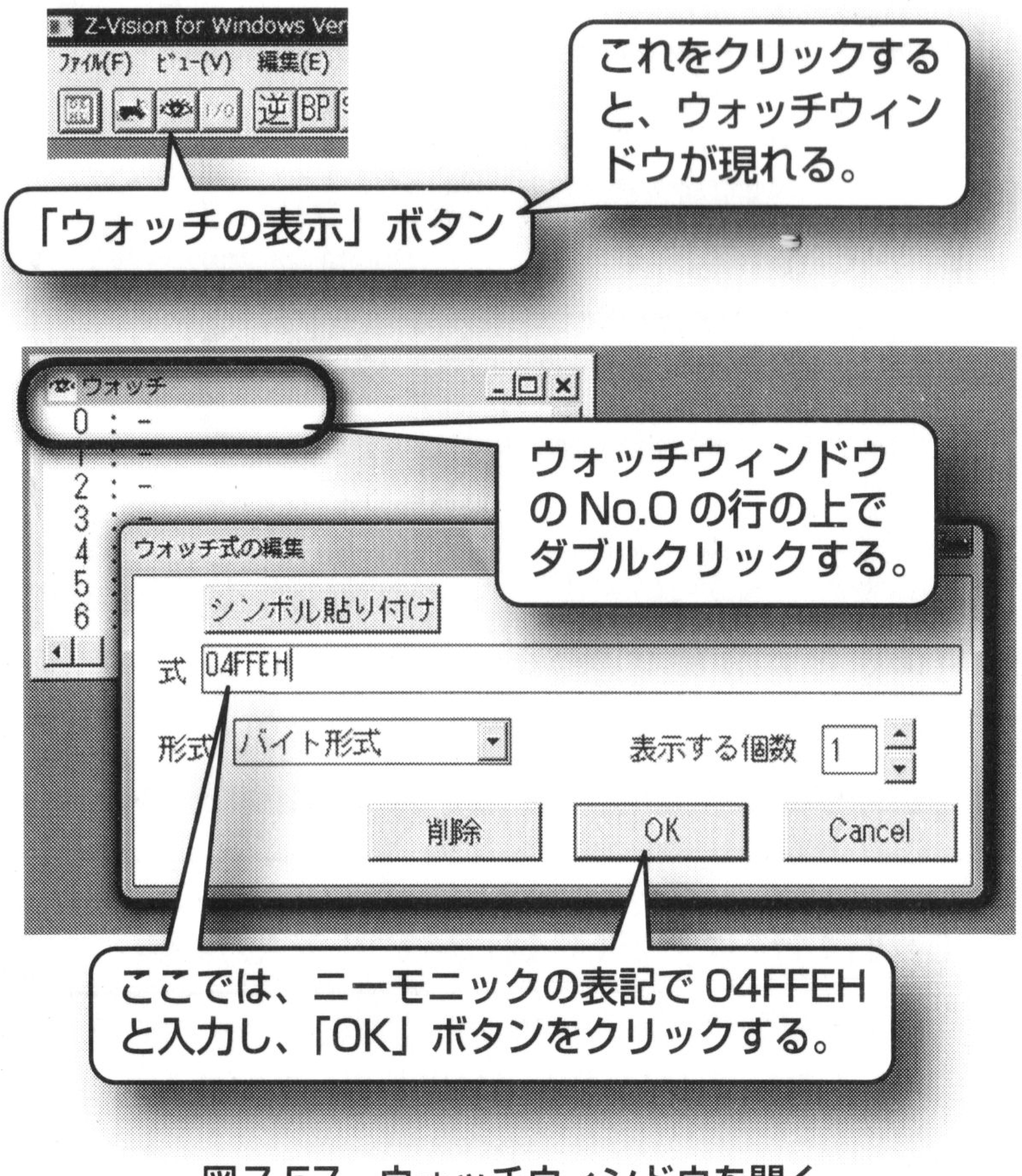


図7-57　ウォッチウィンドウを開く

③LD SP,5000Hを実行した場合、5000Hには変化が起きないはずですが、それを確認するため、このアドレスの内容も表示させます。

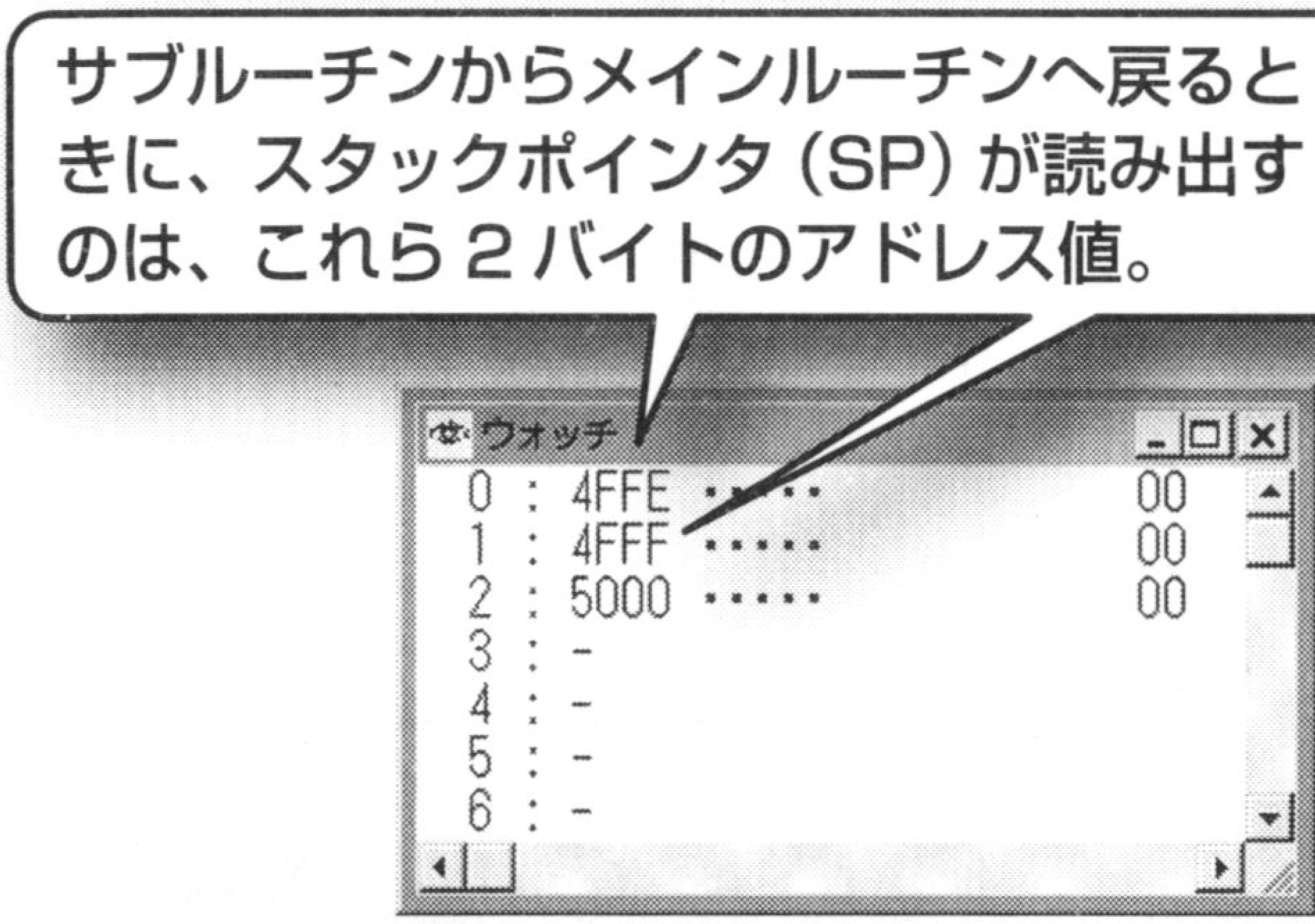


図7-58　スタック付近を常に表示する

④また、I/Oウィンドウについては、No.1だけを利用します。I/Oアドレスは34Hとしました。

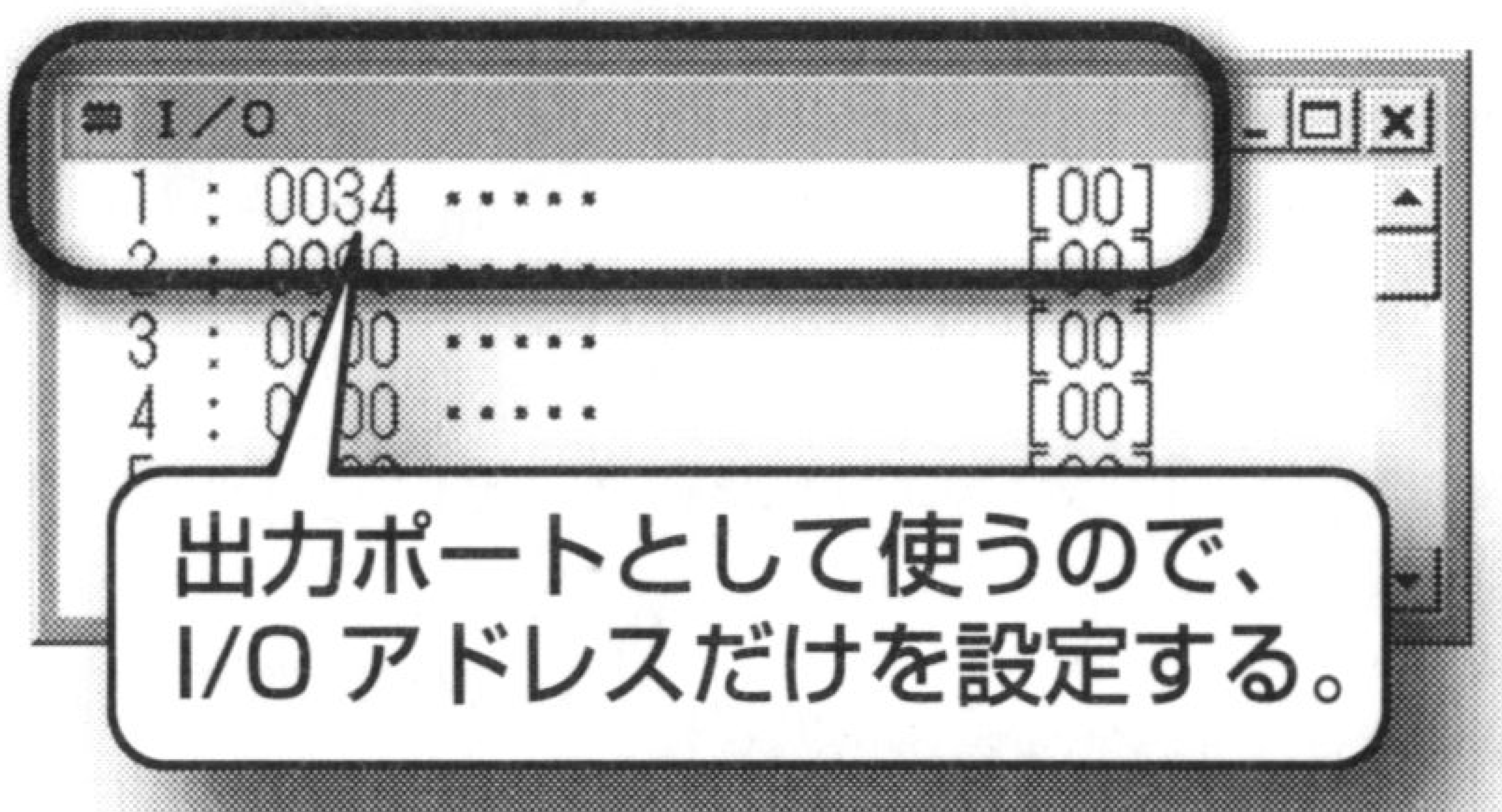


図7-59　出力ポートを設定する

⑤以下、命令文を1行ずつ実行した様子です。ウィンドウの数が多いので、画面上の近い位置にレイアウトすると見やすくなります。

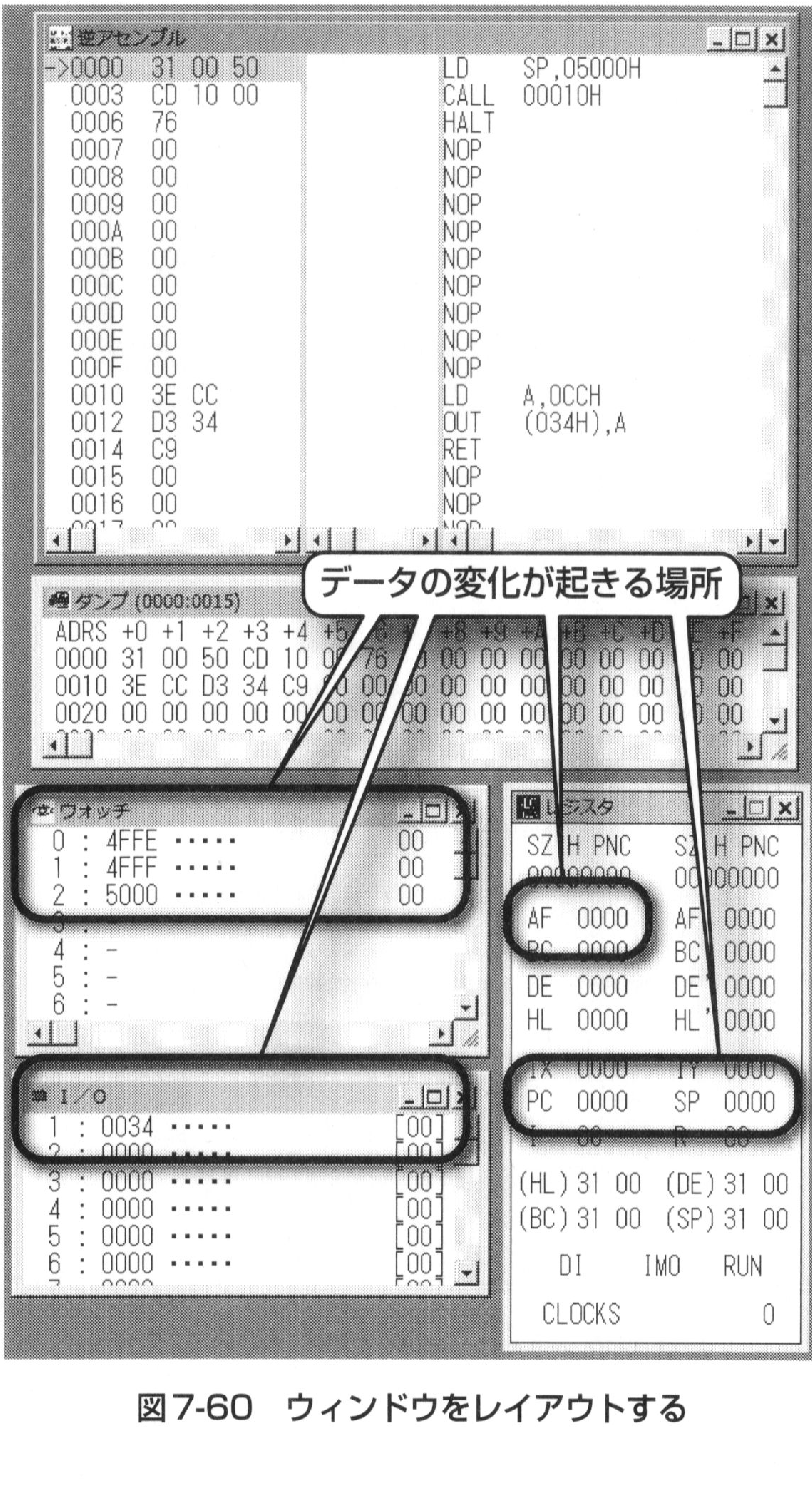


図7-60　ウィンドウをレイアウトする

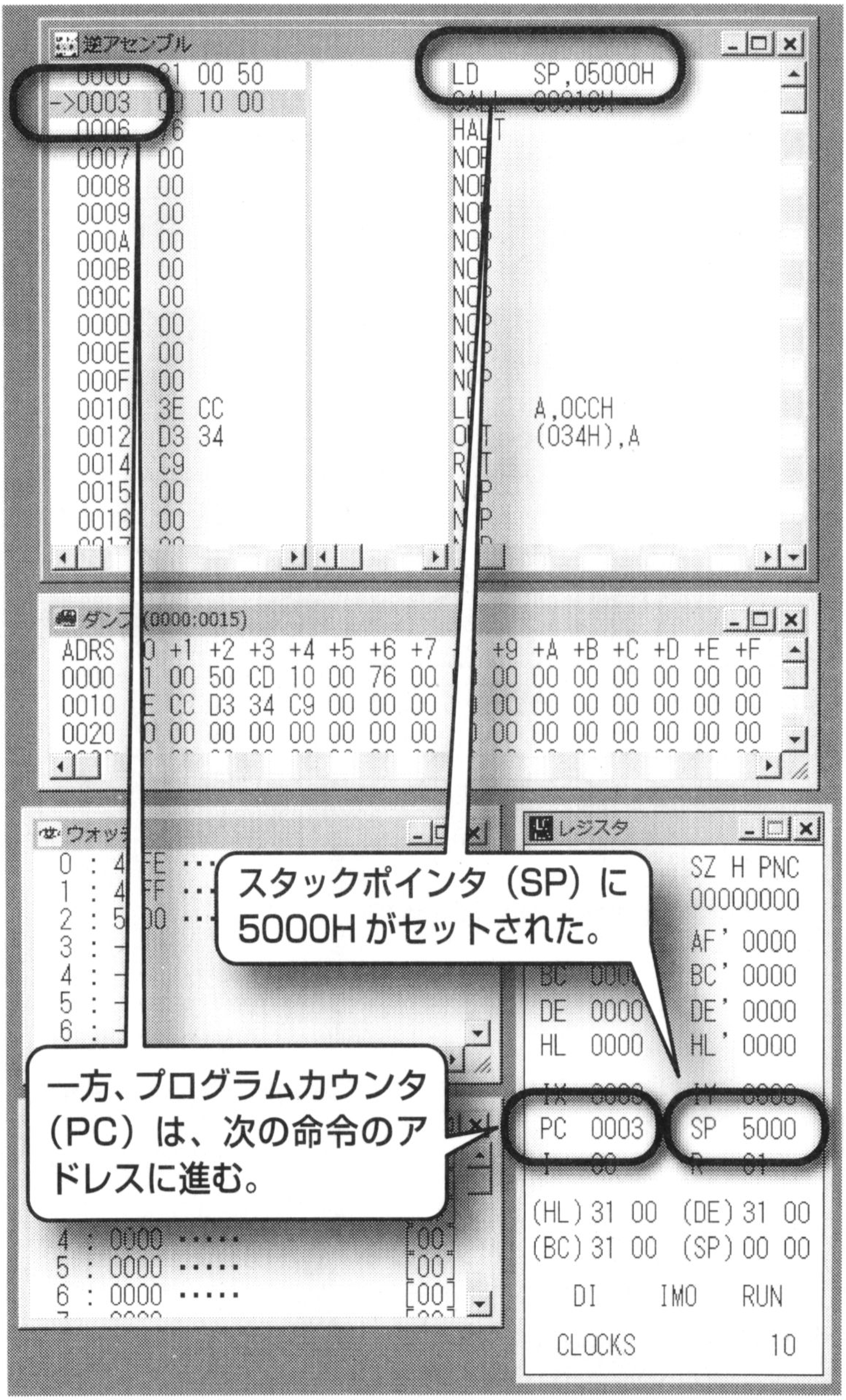


図7-61　LD　SP,5000Hを実行

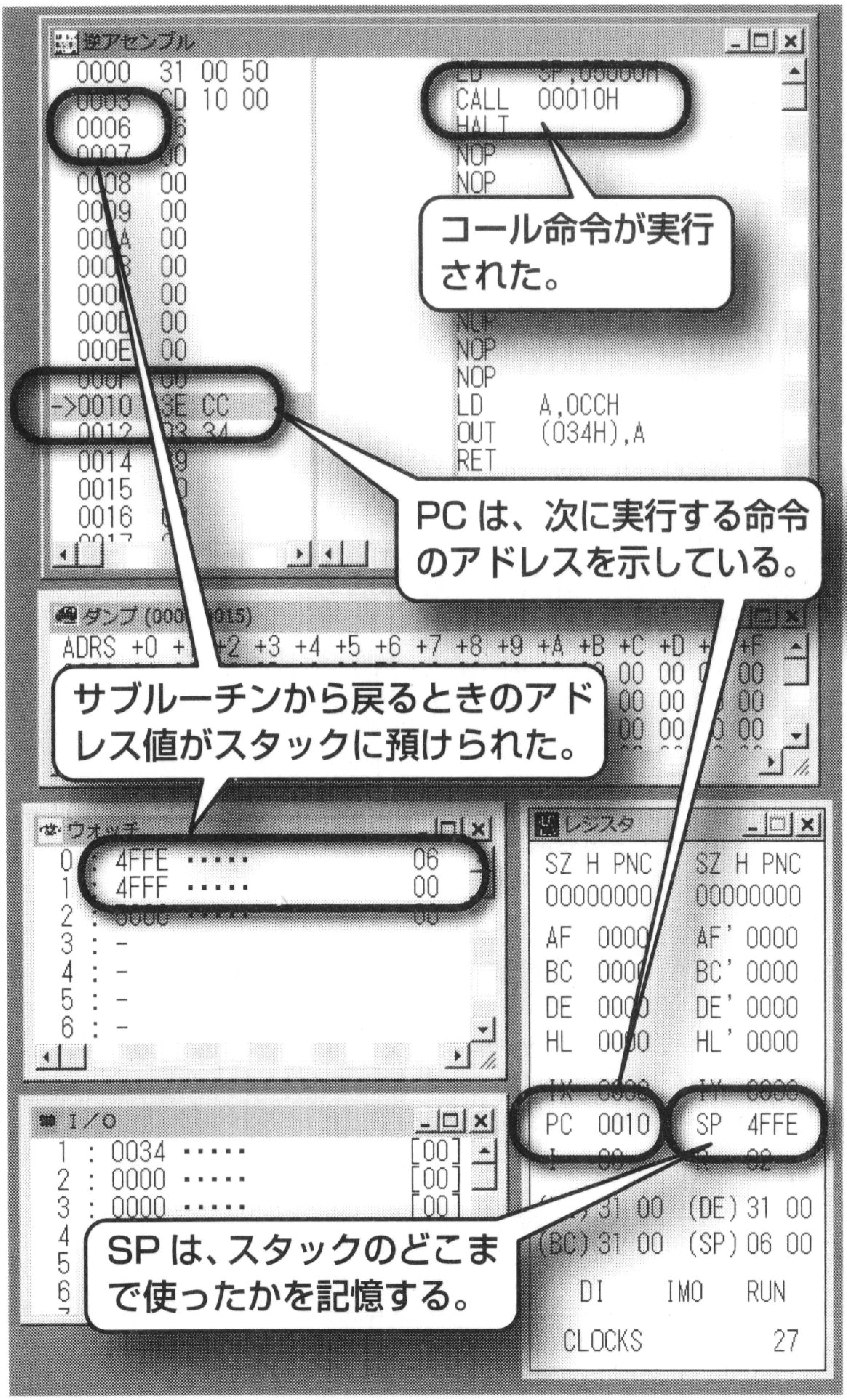


図7-62　CALL　0010Hを実行

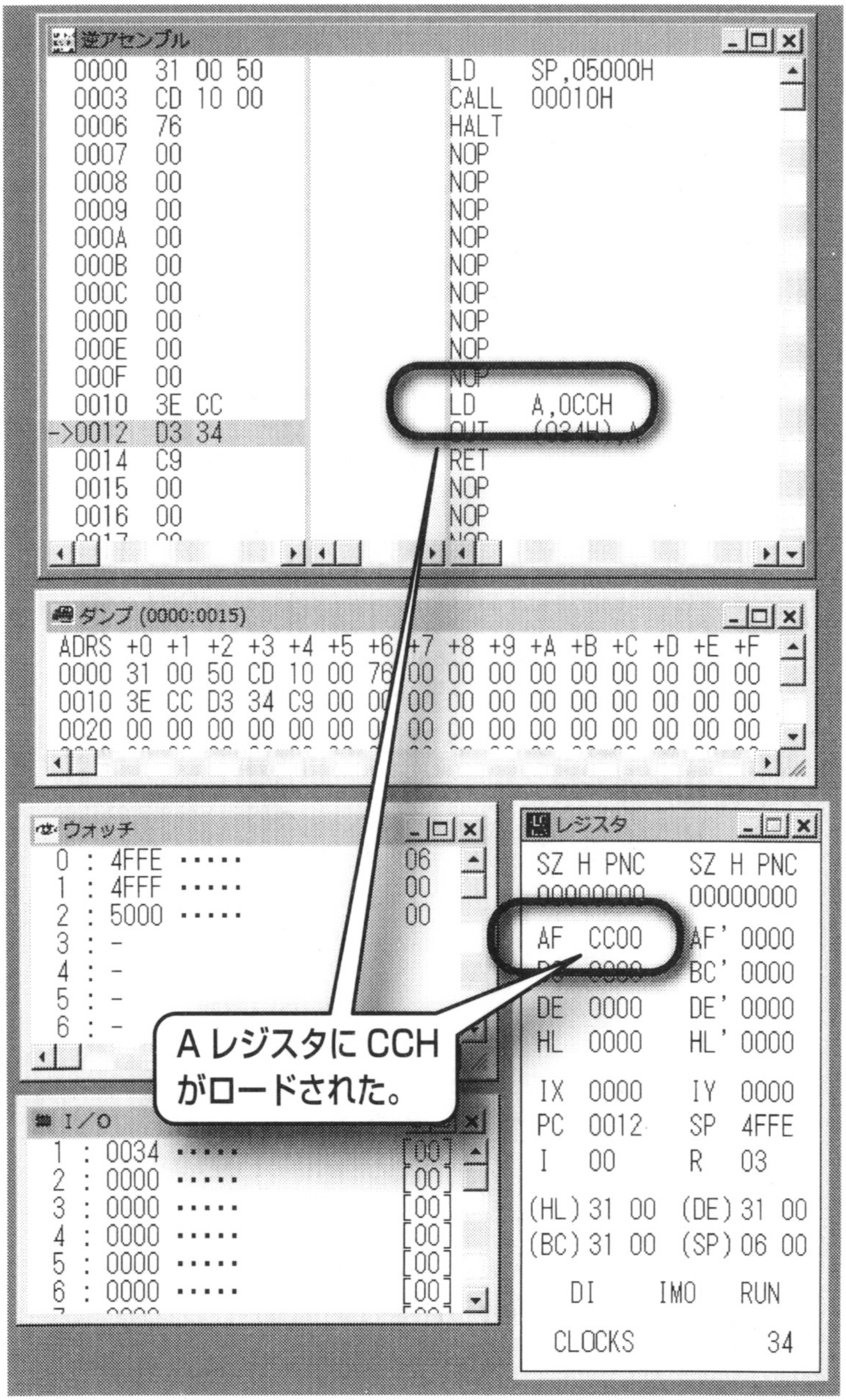


図7-63　LD　A,0CCHを実行

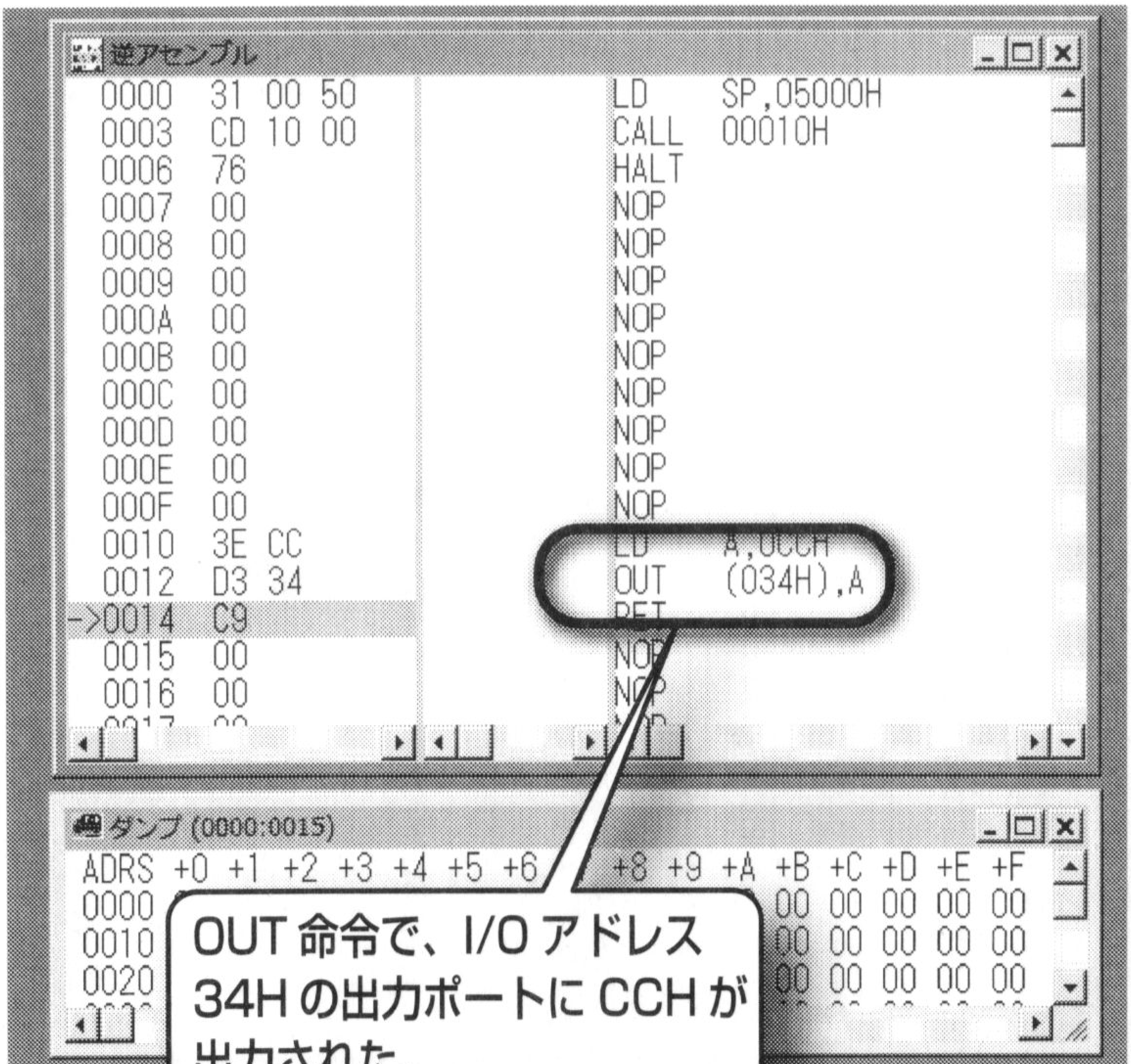


図7-64　サブルーチンのOUT命令を実行

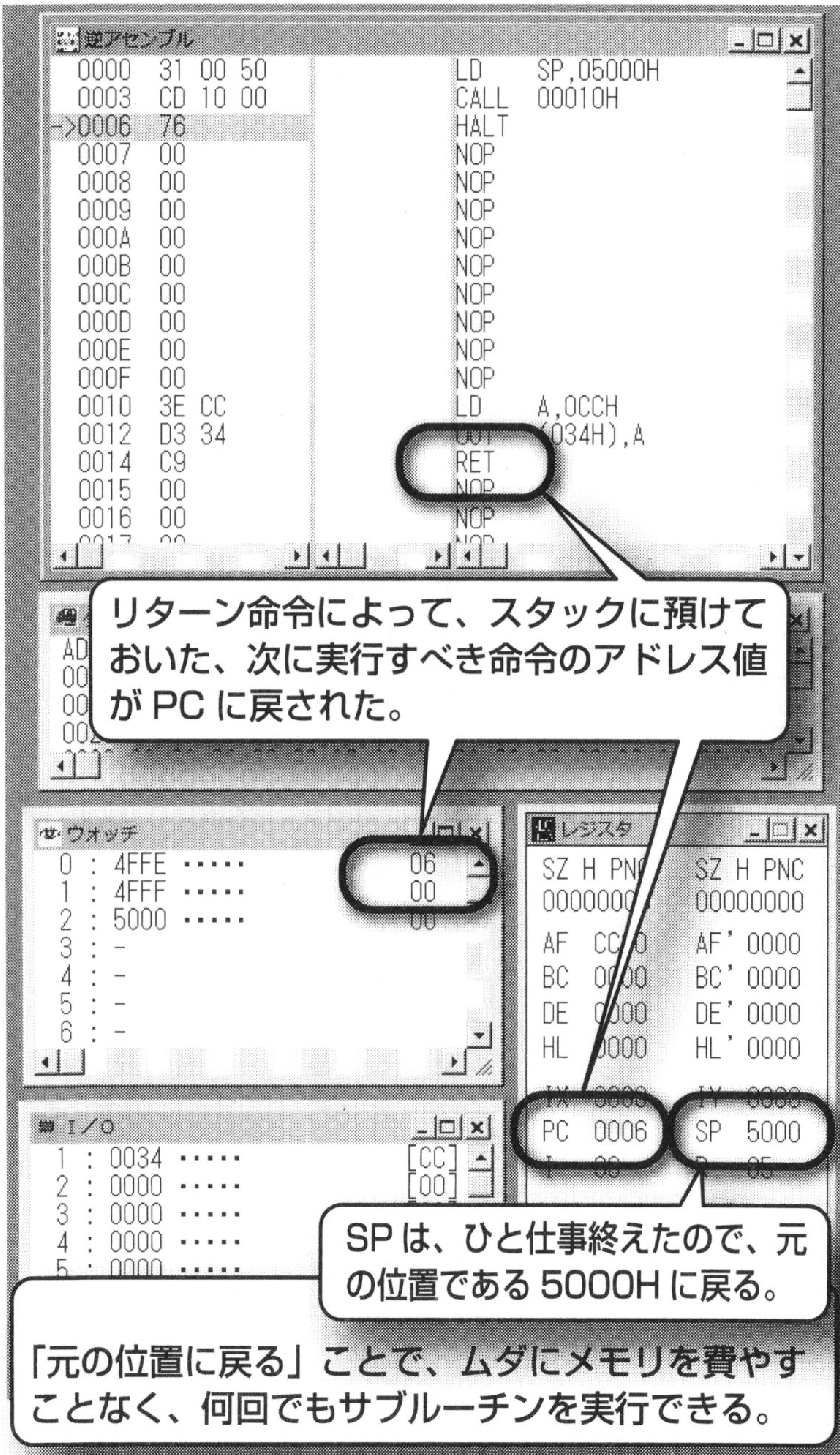


図7-65　RET命令を実行

⑥ところで、ウォッチウィンドウにアドレスを登録したあとは、Z-Visionを再起動しても、削除の操作をしないと、再び同じ状態で現れます。

図7-66の操作によって、指定したアドレスと数値を削除することができます。

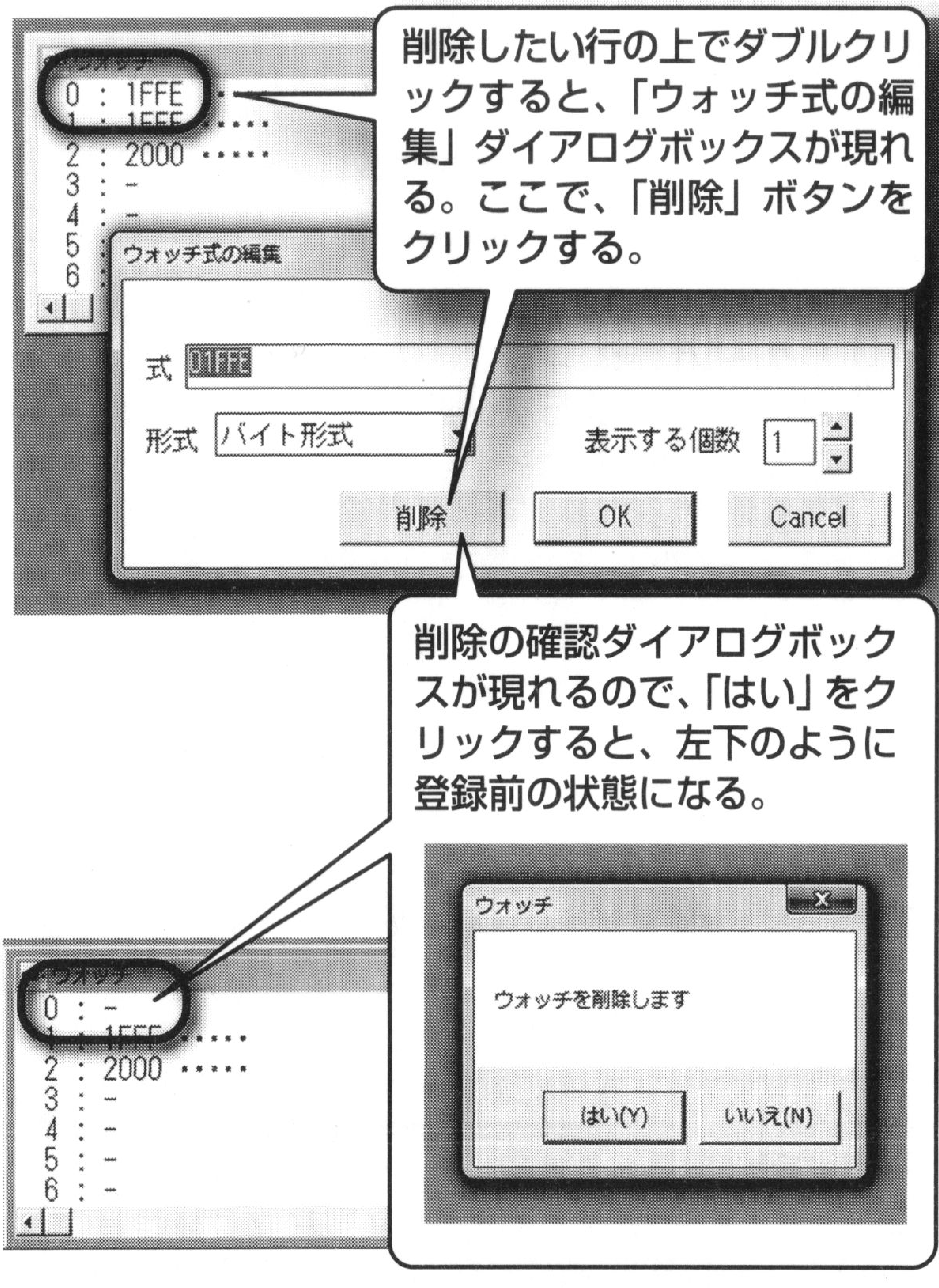


図7-66　ウォッチの削除

7.11　プッシュ/ポップ命令

プッシュ動作とポップ動作については、すでに7.10節で述べましたが、これらの言葉をそのままOPコードにした命令があります。

PUSH命令とPOP命令は、ある特定のレジスタペアを複数の目的に用いるときに実行します。

たとえば、プログラムの途中にAFレジスタペアが別の目的で必要になったときに、PUSH AFを実行すると、CPUは次のような一連の操作を、ユーザには見えないところで実行します。

```
DEC   SP      ; SPの値を－1する
LD    (SP),A
DEC   SP      ; SPの値を－1する
LD    (SP),F
```

つまり、コール命令でメインルーチンに戻るときの番地をスタックに預けたように、レジスタペアの値をスタックに預けてくれるわけです。そして、リターン命令に相当するのが、POP AFです。

それでは、プッシュ/ポップ命令を用いた実験をしてみましょう。

少し機械語が長いですが、「31 00 30 3E 11 06 22 0E 33 F5 C5 3E 44 37 06 55 0E 66 C1 F1 76」です。実験の様子は、図7-68～図7-80に示しました。

このプログラムでは、AFとBCの２つのレジスタペアをプッシュ/ポップしています。２組あるPUSHとPOPの順序に注意してください。

load register pair ss onto stack

PUSH ss

ss レジスタペアの内容をスタック領域上にロードする。ss の上位 1 バイトは SP−1 が示すアドレスにロードされ、ss の下位 1 バイトは SP−2 が示すアドレスにロードされる。

◆フラグは変化しない。

ss→	BC	DE	BC	DE
機械語	C5	D5	E5	F5

load register pair ss with top of stack

POP ss

スタック領域の内容を ss レジスタペアにロードする。SP−1 が示すアドレスの値は ss の上位 1 バイトにロードされ、SP−2 が示すアドレスの値は ss の下位 1 バイトにロードされる。

◆フラグは変化しない。

ss→	BC	DE	BC	DE
機械語	C1	D1	E1	F1

図7-67　PUSH命令とPOP命令

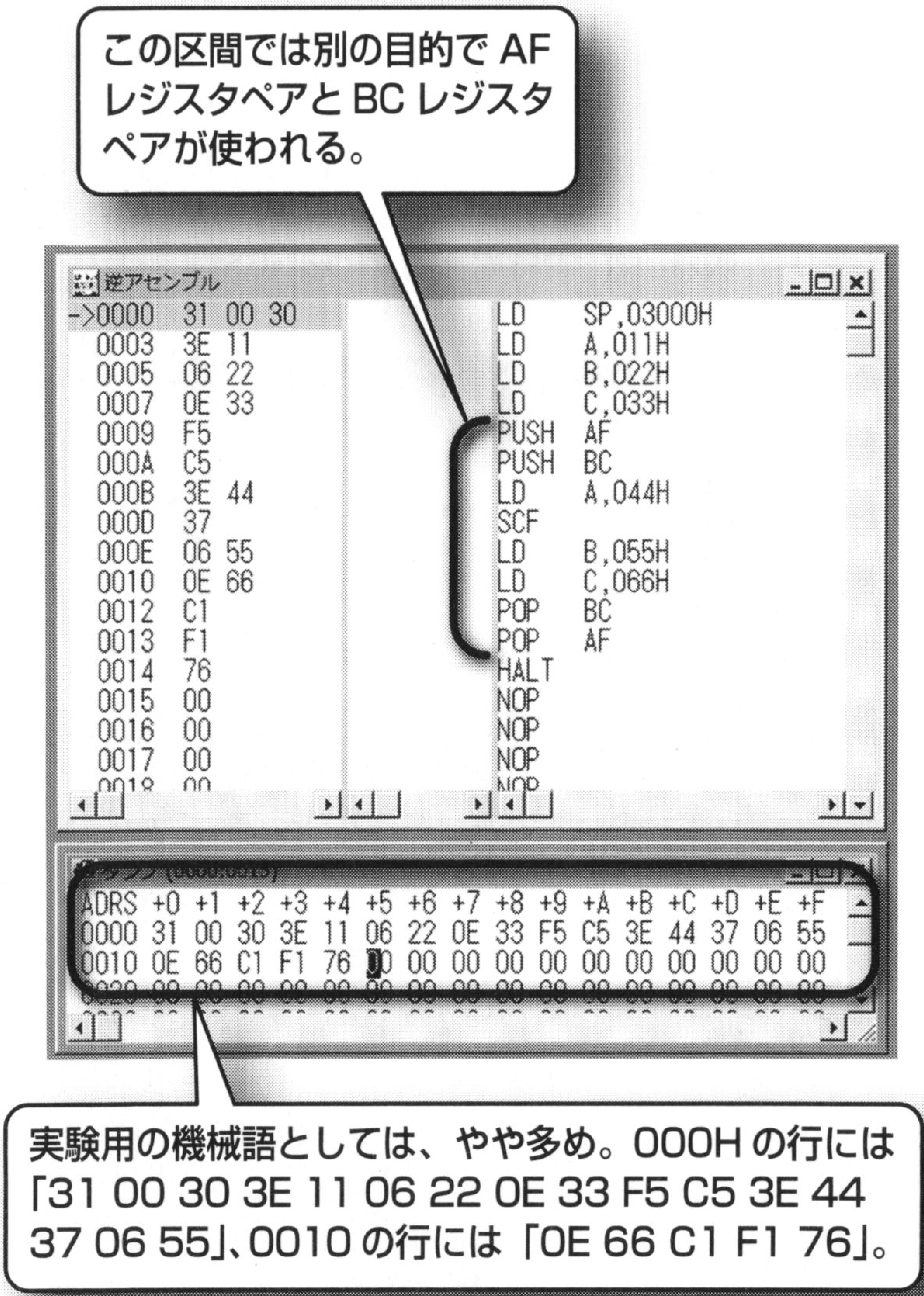


図7-68　PUSH/POP命令を用いた実験

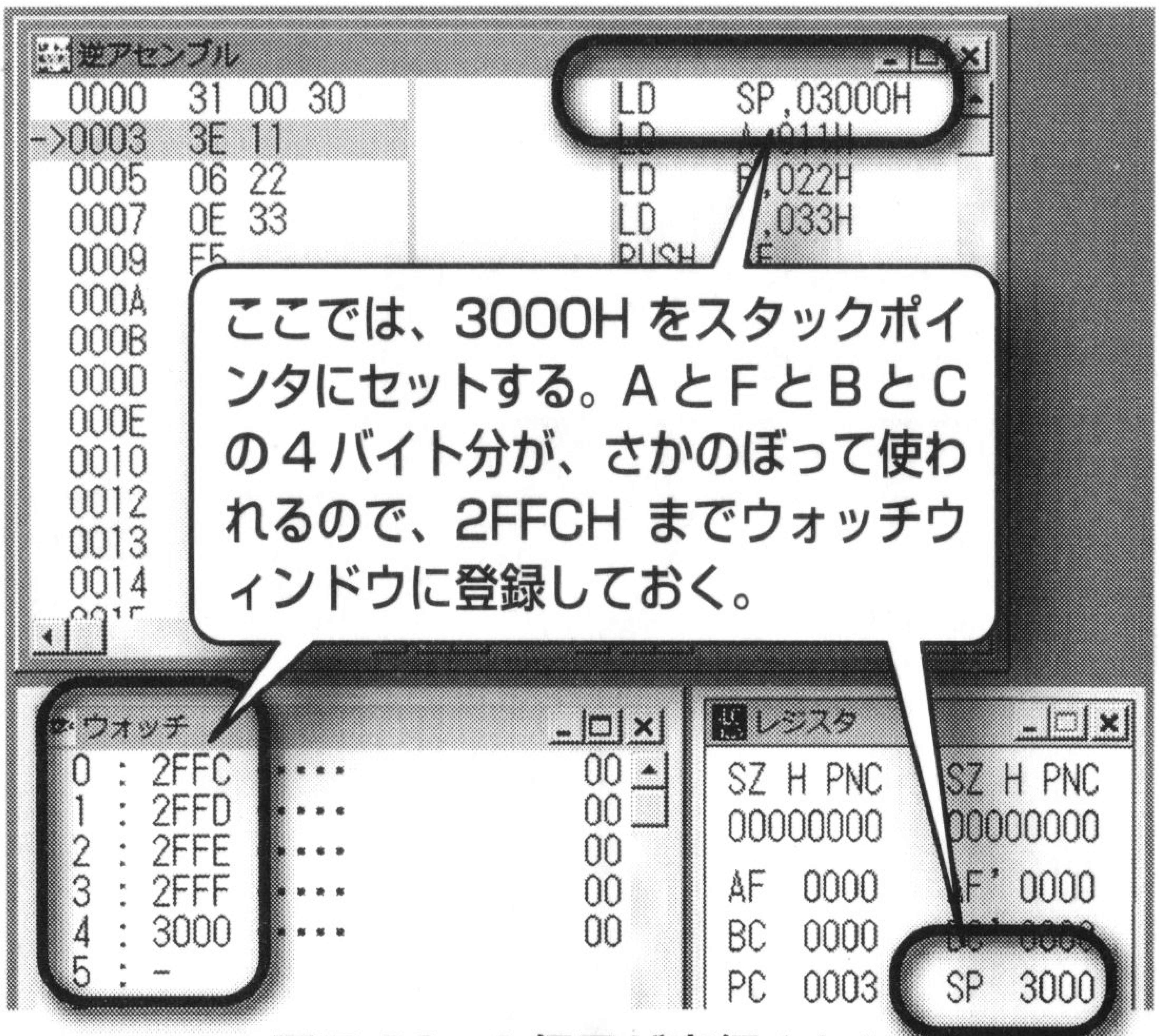


図7-69　1行目が実行された



図7-70　2行目が実行された

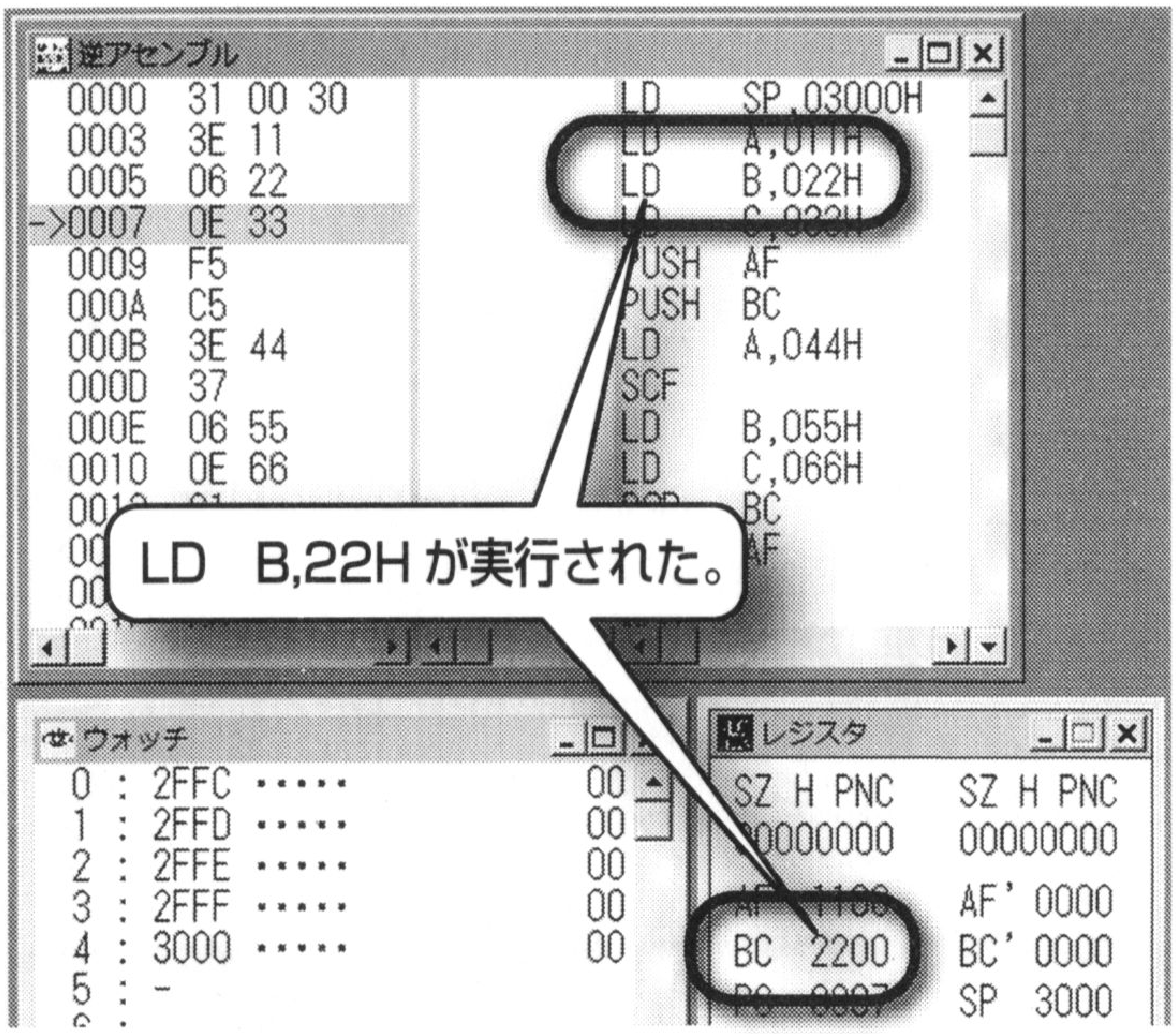


図7-71　3行目が実行された

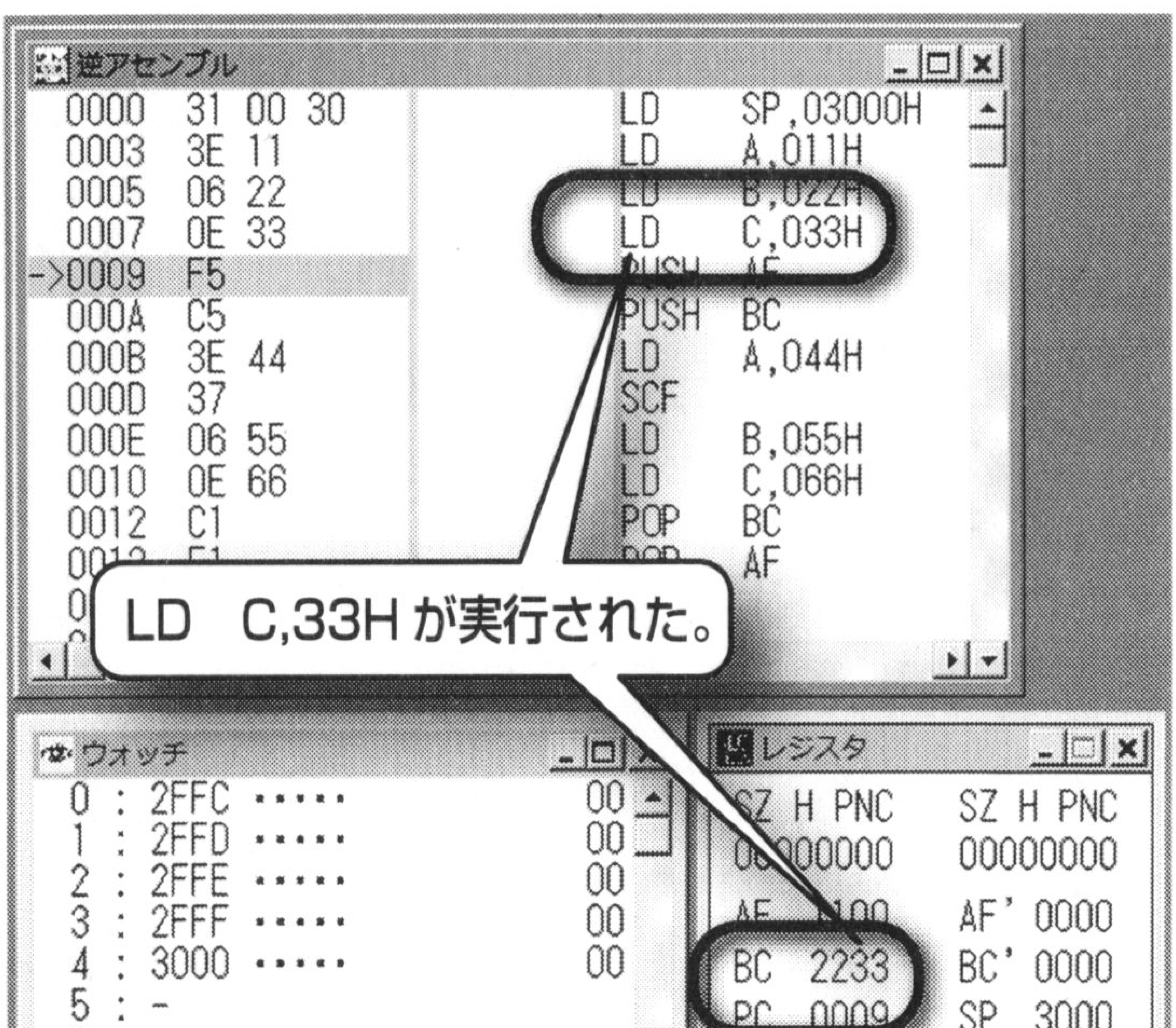


図7-72　4行目が実行された



図7-73　5行目が実行された

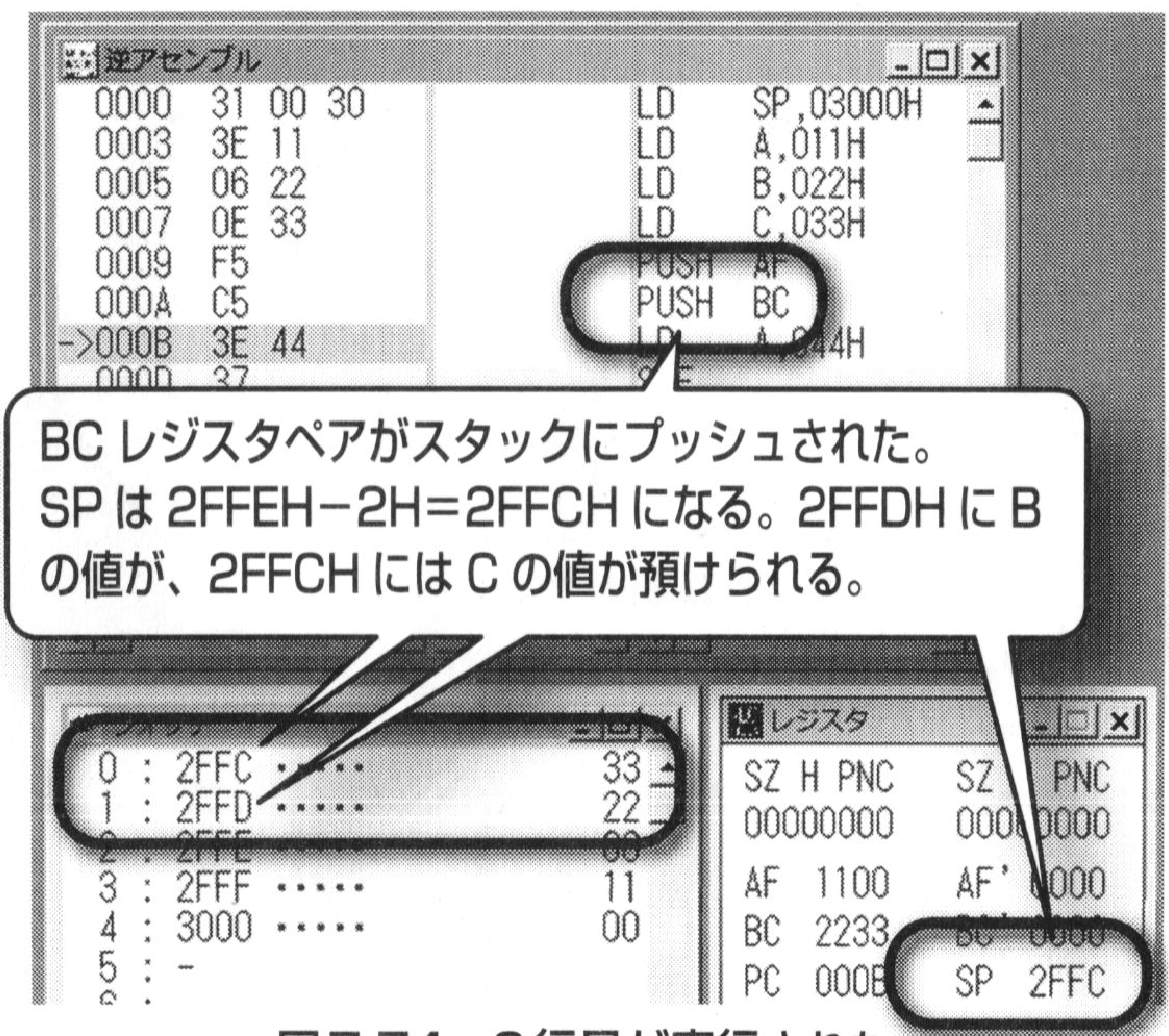


図7-74　6行目が実行された

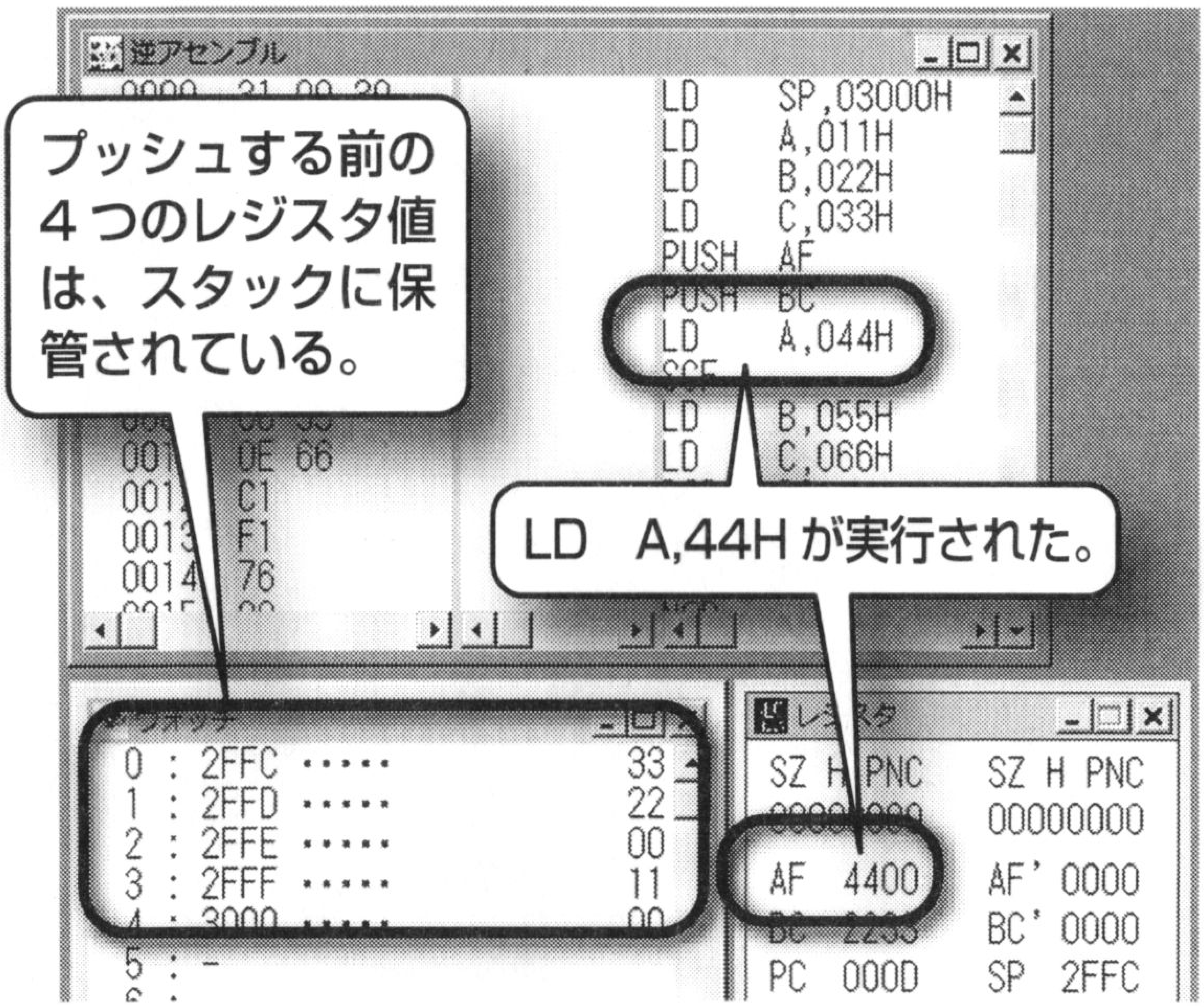


図7-75　7行目が実行された

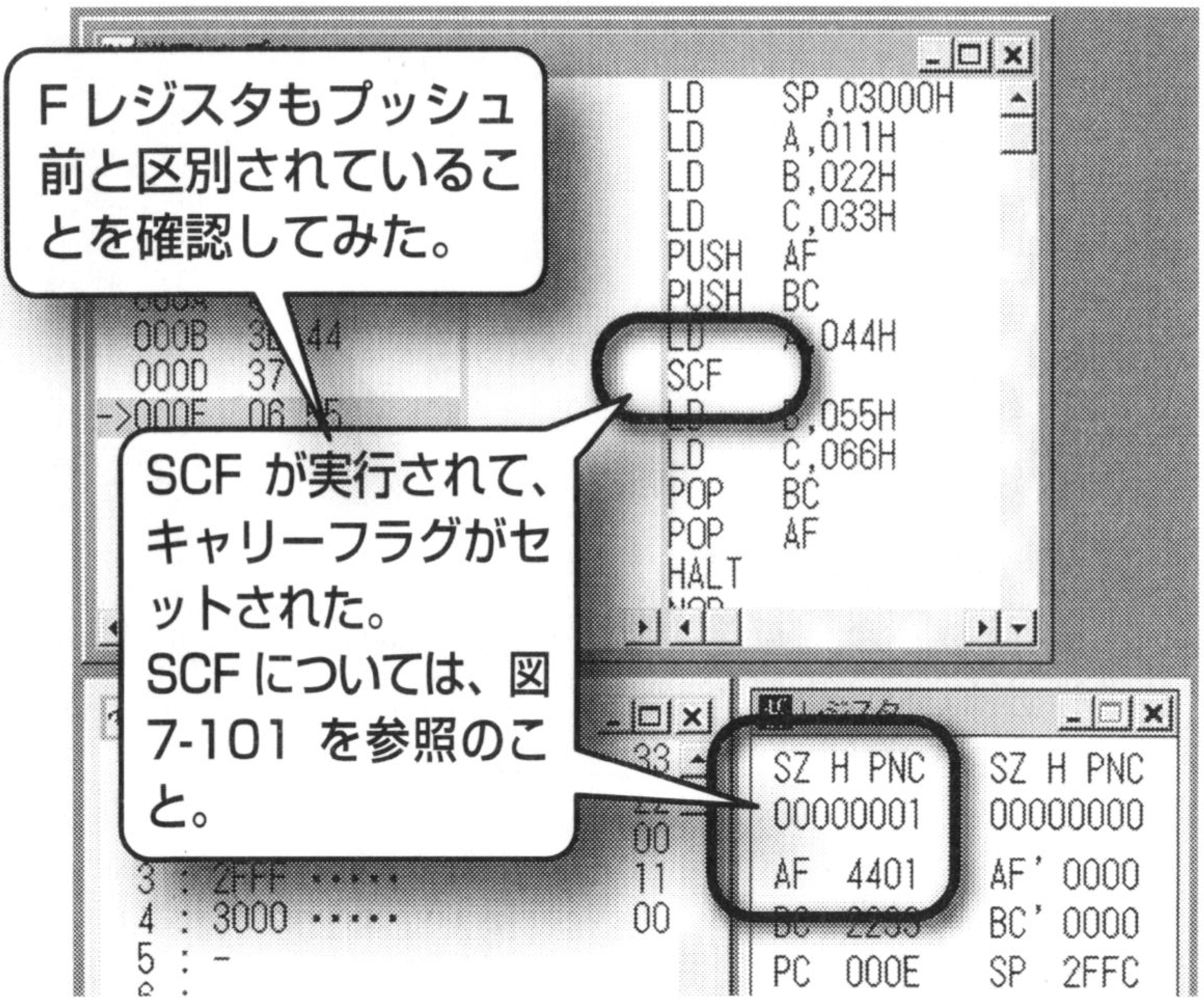


図7-76　8行目が実行された

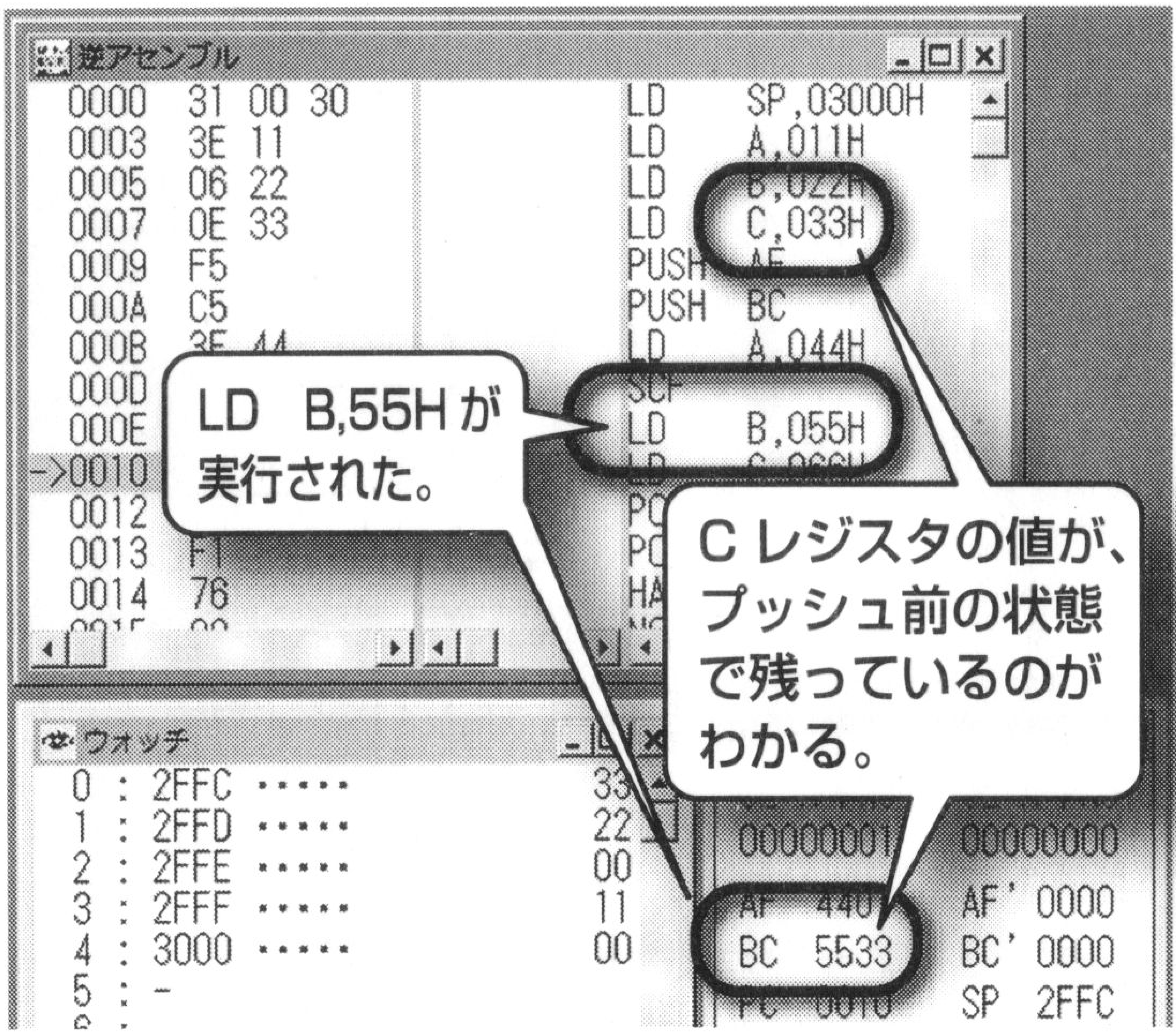


図7-77　9行目が実行された

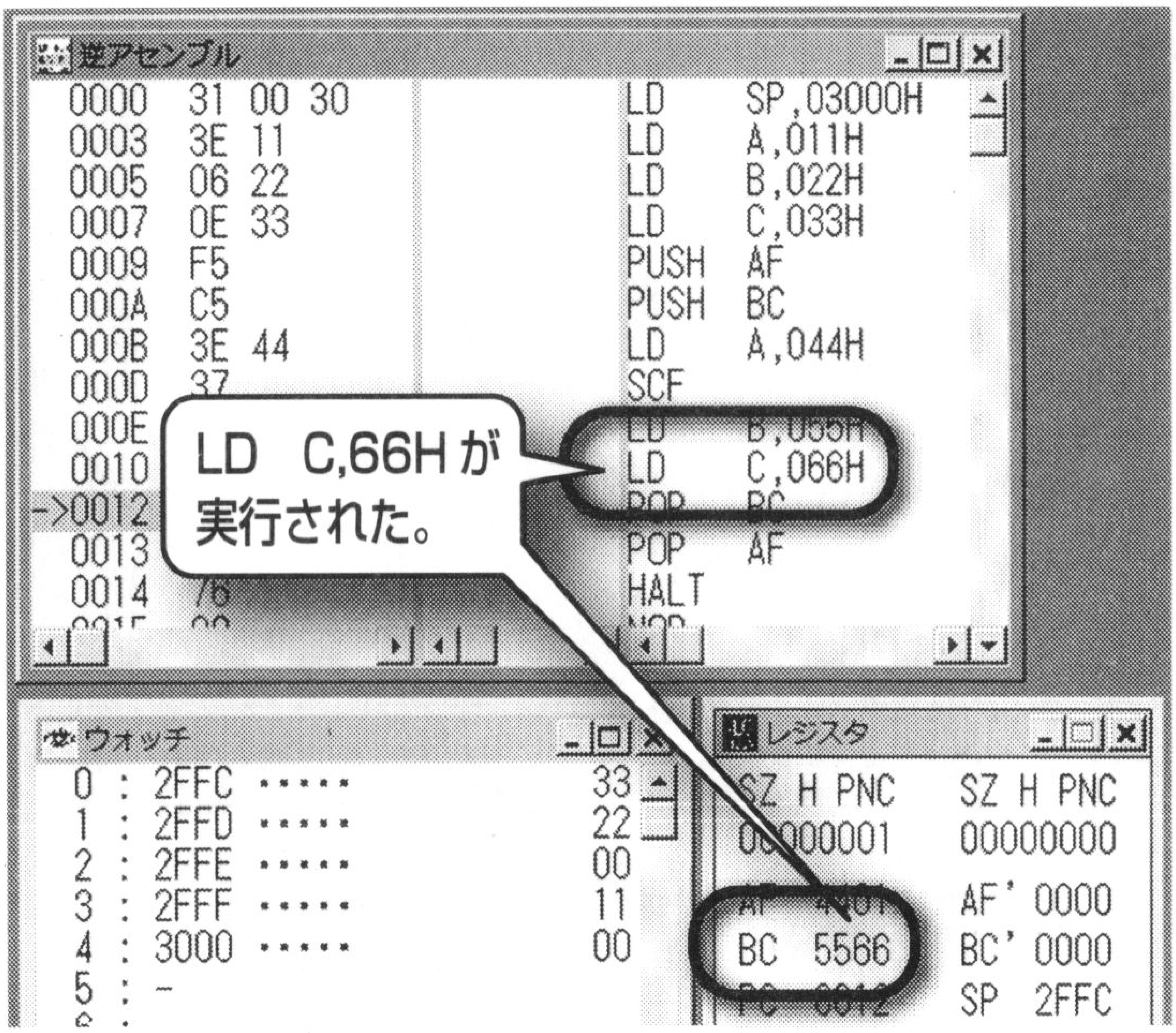


図7-78　10行目が実行された

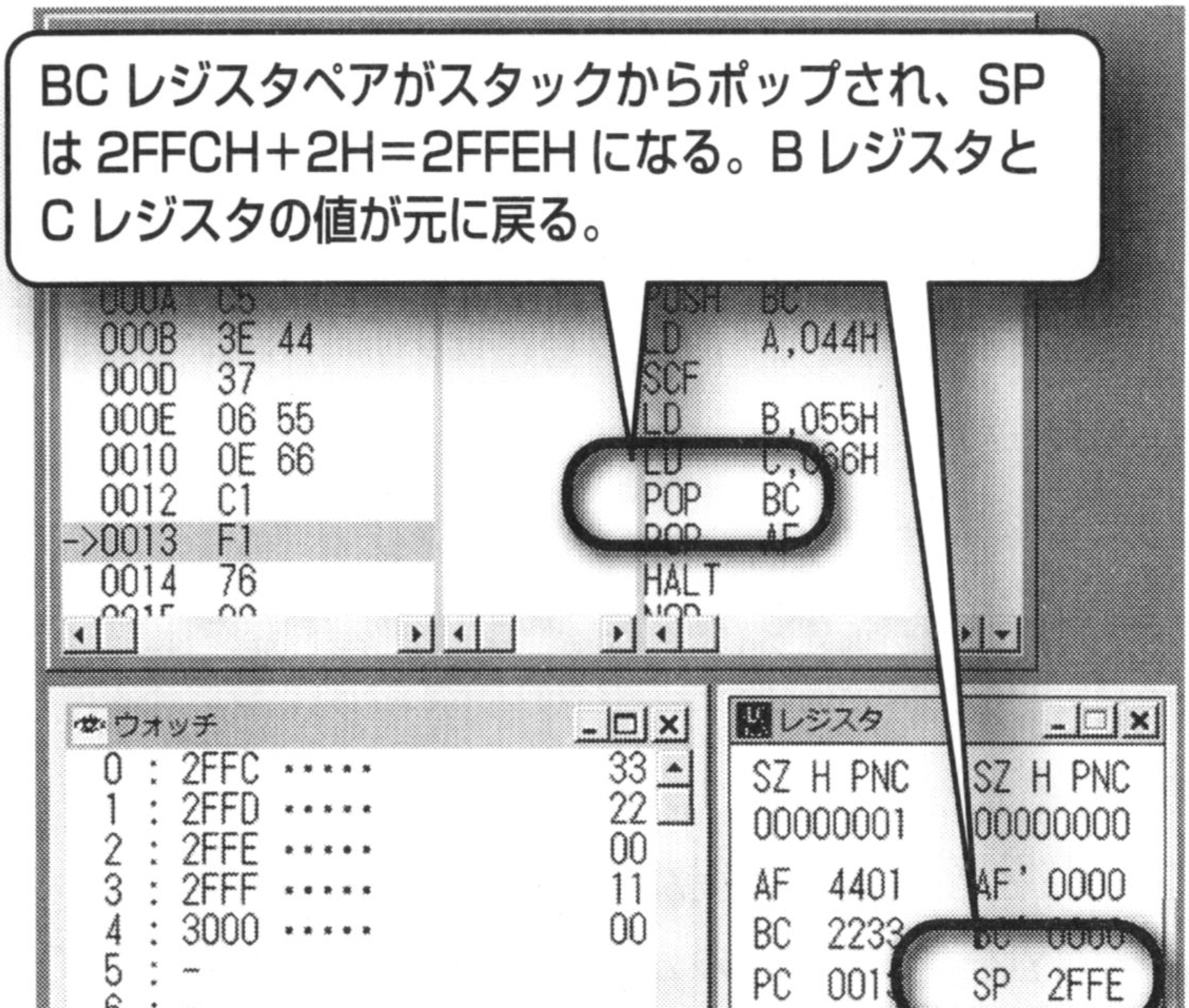


図7-79　11行目が実行された

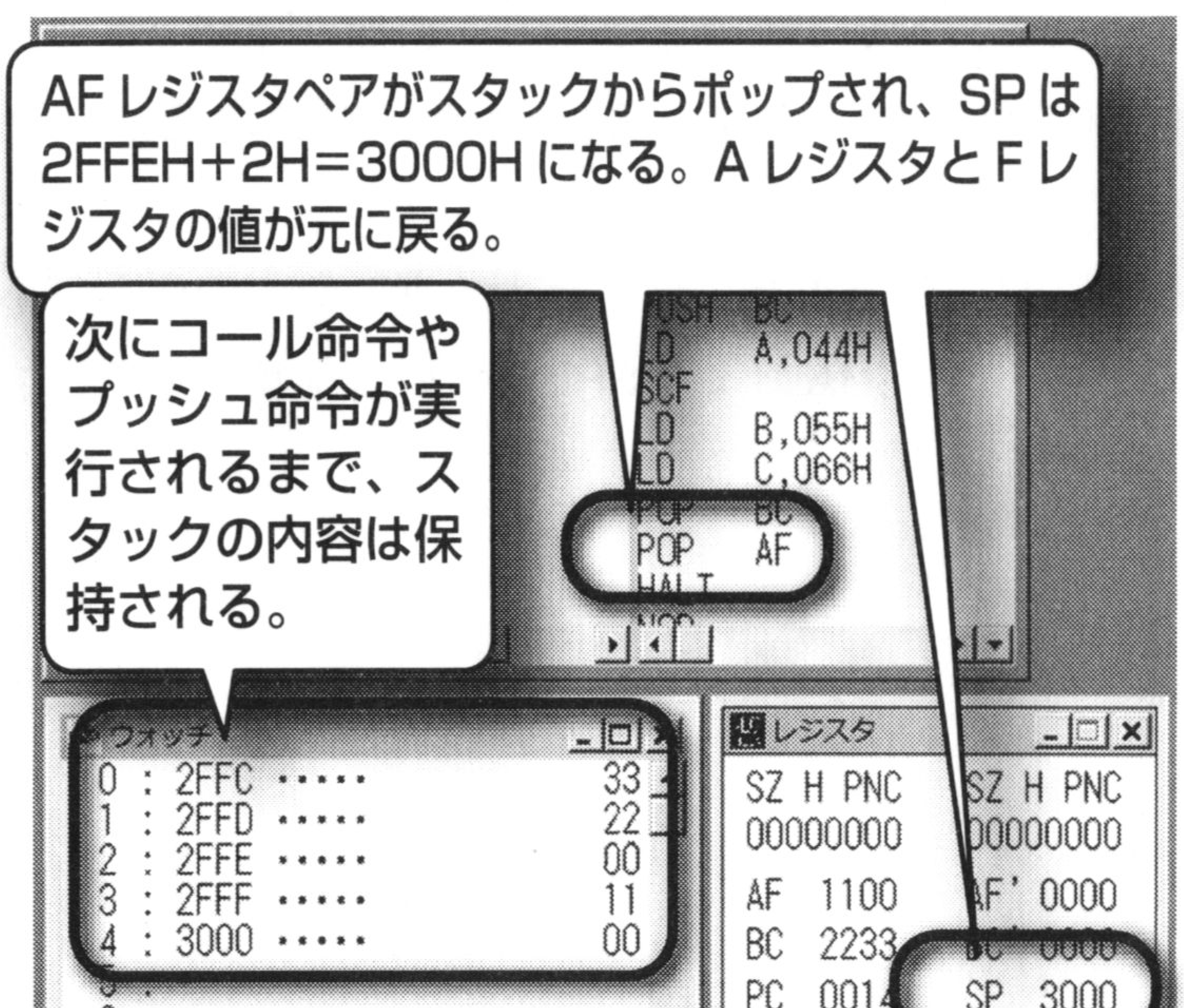


図7-80　12行目が実行された

7.12 ローテート/シフト命令

スポーツのバレーボールでは、サーブ権を得るたびに各選手が時計回りに位置を変えます。

これをローテーションと呼び、6人が1つの輪になっているわけですが、レジスタの各ビットにもこれと同様な動作をさせることができます。

ローテート命令を使うと、Aレジスタを筆頭に、B、C、D、E、H、Lの各レジスタについて、ビットのローテーションが可能です。

CY（キャリーフラグ）を含めてローテートするか、含めないかを区別しているため、命令の種類が多くなっています。CYを含めないのがRLC、RRC、RLCA、RRCA命令、CYを含めるのがRL、RR、RLA、RRAの4命令です。

そのほか、Aレジスタとメモリとの間で4ビットずつをまとめてローテートするRLD命令とRRD命令があります。

一方、ローテートせずにビットを押しだすのがシフト命令です。SLA、SRAおよびSRL命令の3種類があります。

RLC (HL)　CB 06

HLレジスタペアが示すアドレスのメモリの内容を1ビットだけ左にシフトする。ビット7は、ビット0とキャリーフラグに入る。

CY ← 7 6 5 4 3 2 1 0 ←（ビット7 → ビット0）

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

rotate left location (HL) circular

パリティフラグ

rotate left register r circular

RLC r

rレジスタの内容を1ビットだけ左にシフトする。ビット7は、ビット0とキャリーフラグに入る。

CY ← 7 6 5 4 3 2 1 0 ←（ビット7 → ビット0）

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

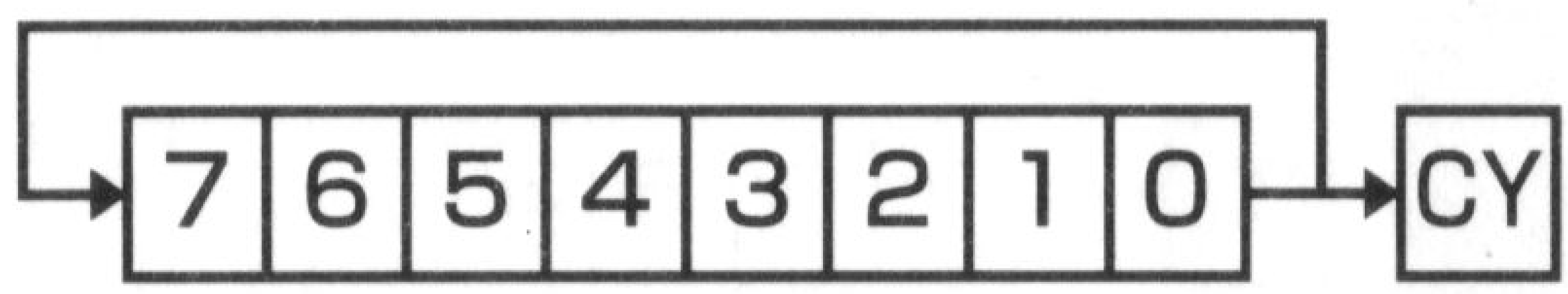
r→	A	B	C	D	E	H	L
機械語	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05

パリティフラグ

図7-81　RLC命令

RRC (HL)　　CB 0E

HLレジスタペアが示すアドレスのメモリの内容を1ビットだけ右にシフトする。ビット0は、ビット7とキャリーフラグに入る。



S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

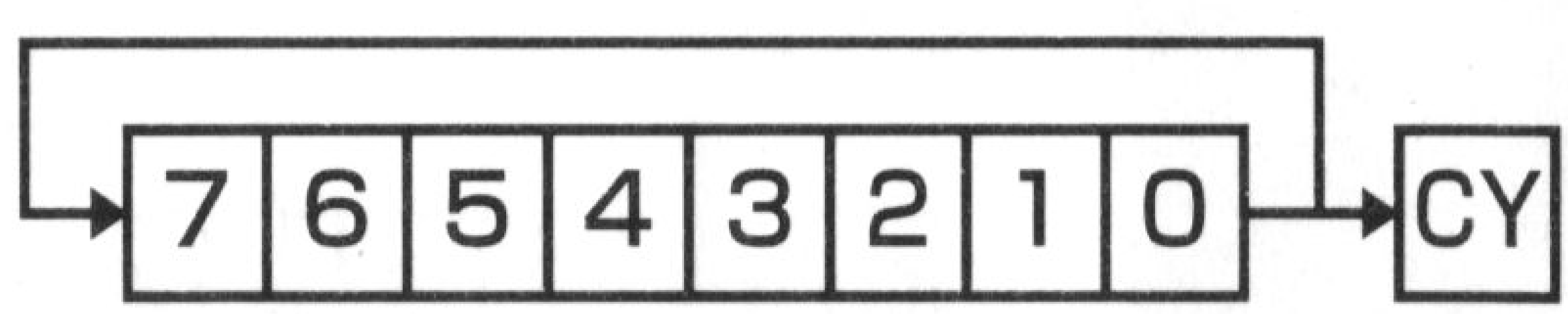
rotate right location (HL) circular

パリティフラグ

rotate right register r circular

RRC r

rレジスタの内容を1ビットだけ右にシフトする。ビット0は、ビット7とキャリーフラグに入る。



S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D

パリティフラグ

図7-82　RRC命令

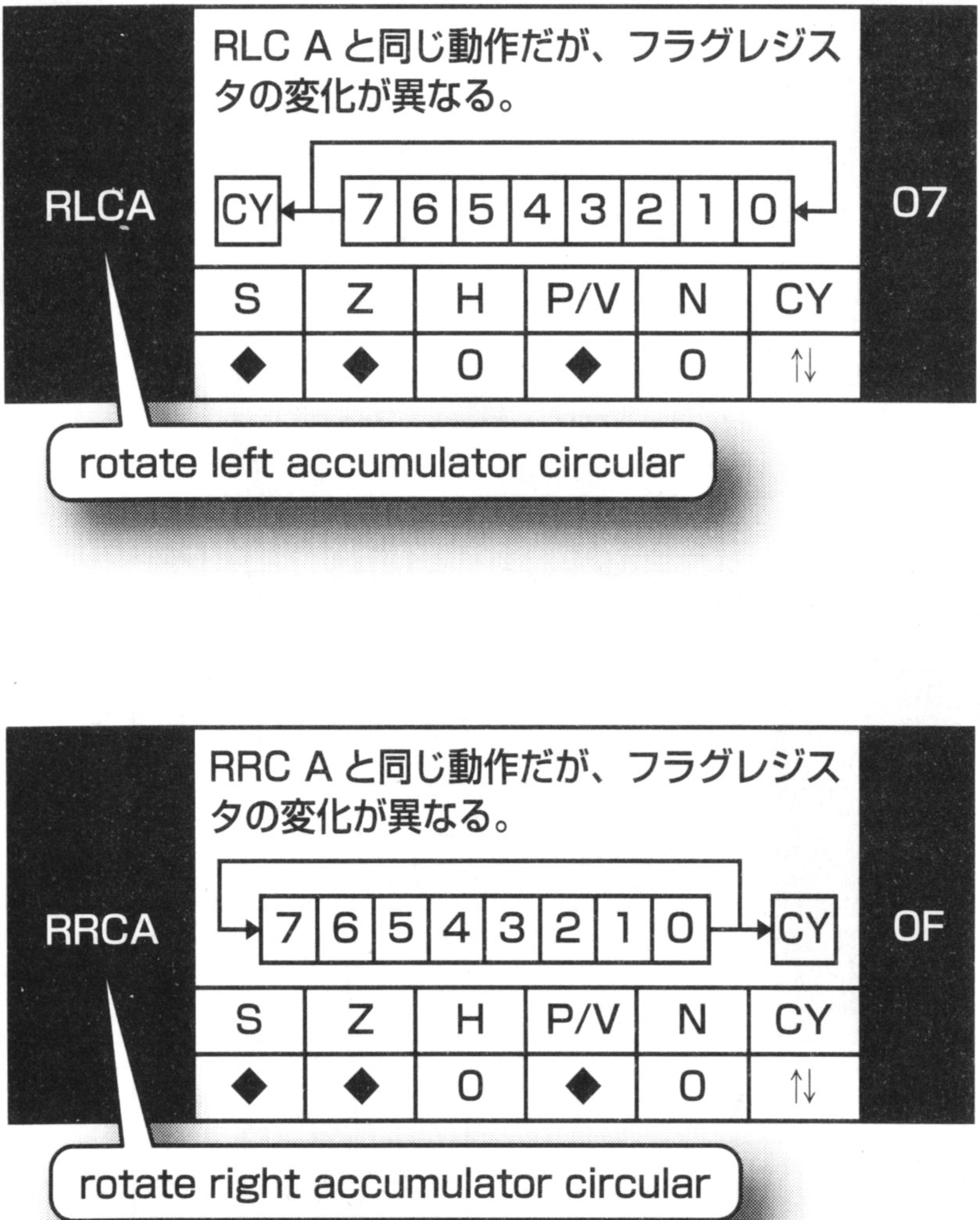
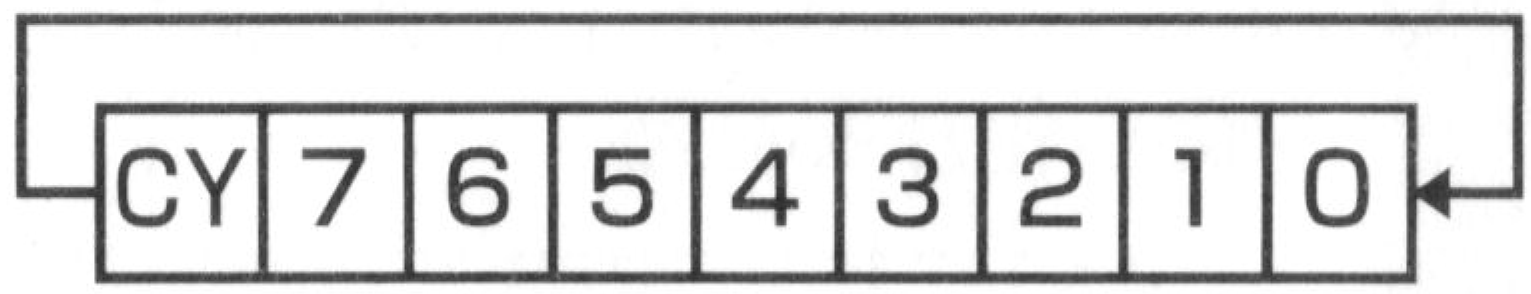


図7-83　RLCA命令とRRCA命令

RL (HL)

HL レジスタペアが示すアドレスのメモリの内容を 1 ビットだけ左にシフトする。キャリーフラグはビット 0 に入る。



CB 16

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

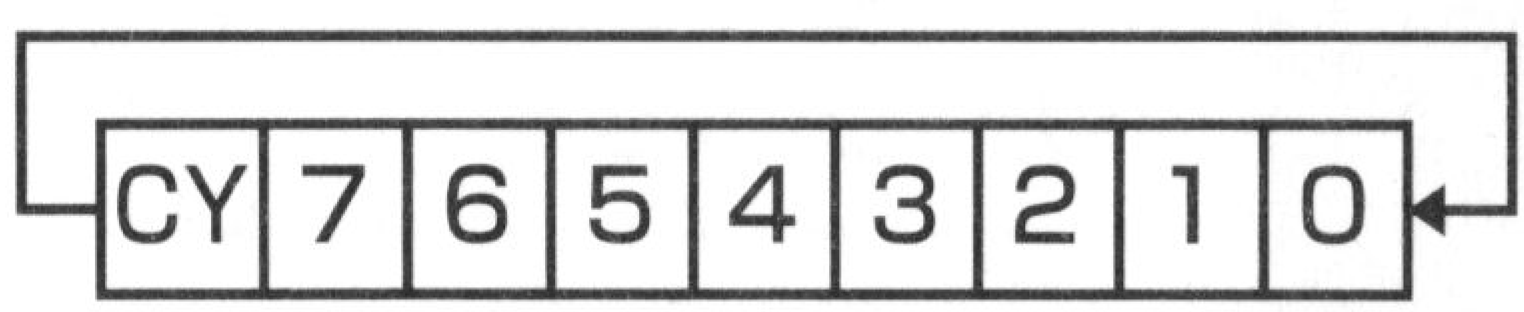
rotate left location (HL) through carry

パリティフラグ

rotate left register r through carry

RL r

r レジスタの内容を 1 ビットだけ左にシフトする。キャリーフラグはビット 0 に入る。



S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15

パリティフラグ

図7-84　RL命令

RR (HL)　　CB 1E

HL レジスタペアが示すアドレスのメモリの内容を 1 ビットだけ右にシフトする。キャリーフラグはビット 7 に入る。

7	6	5	4	3	2	1	0	CY

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

rotate right location (HL) through carry

パリティフラグ

rotate right register r through carry

RR r

r レジスタの内容を 1 ビットだけ右にシフトする。キャリーフラグはビット 7 に入る。

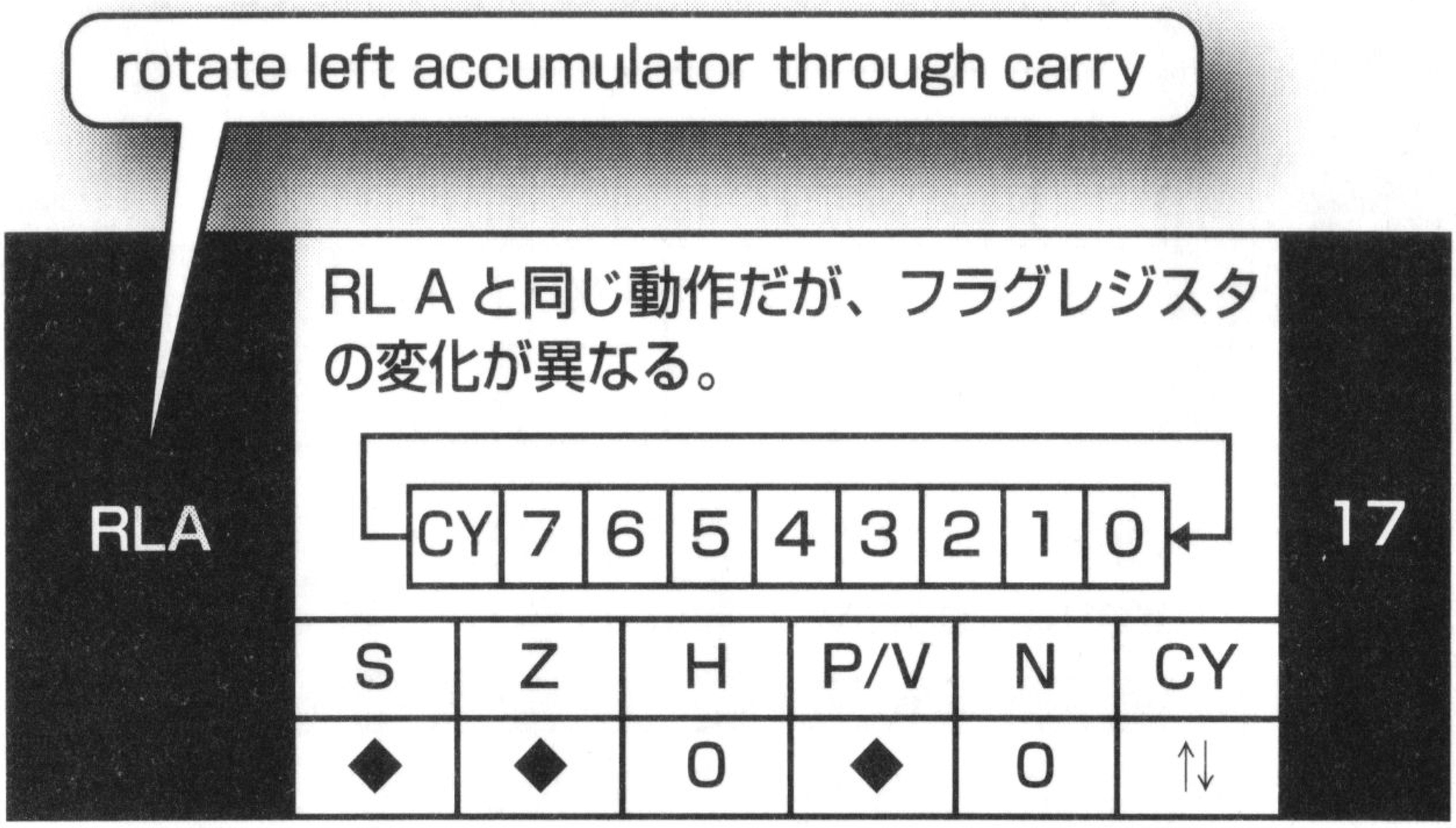
7	6	5	4	3	2	1	0	CY

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D

パリティフラグ

図7-85　RR命令



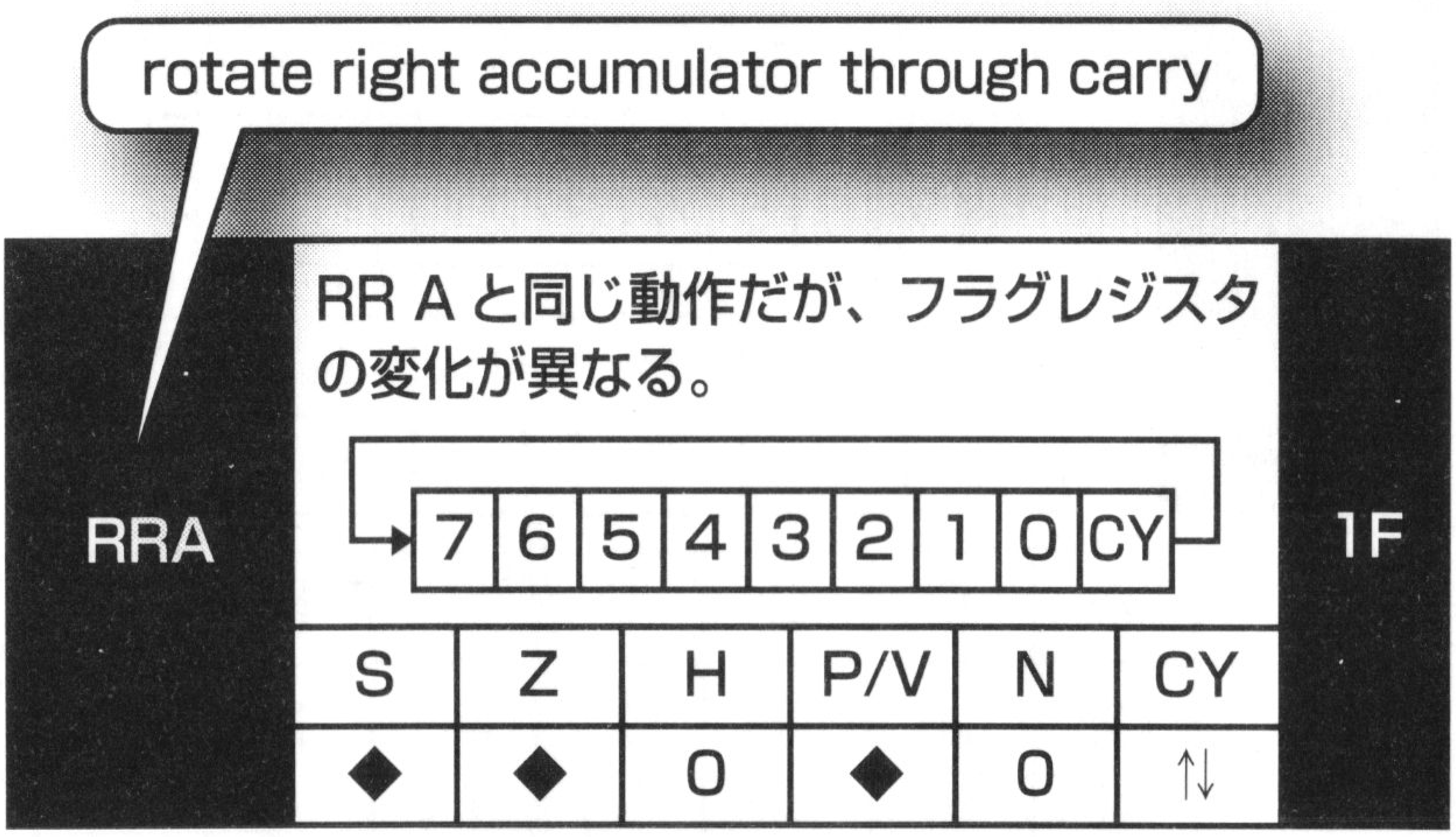


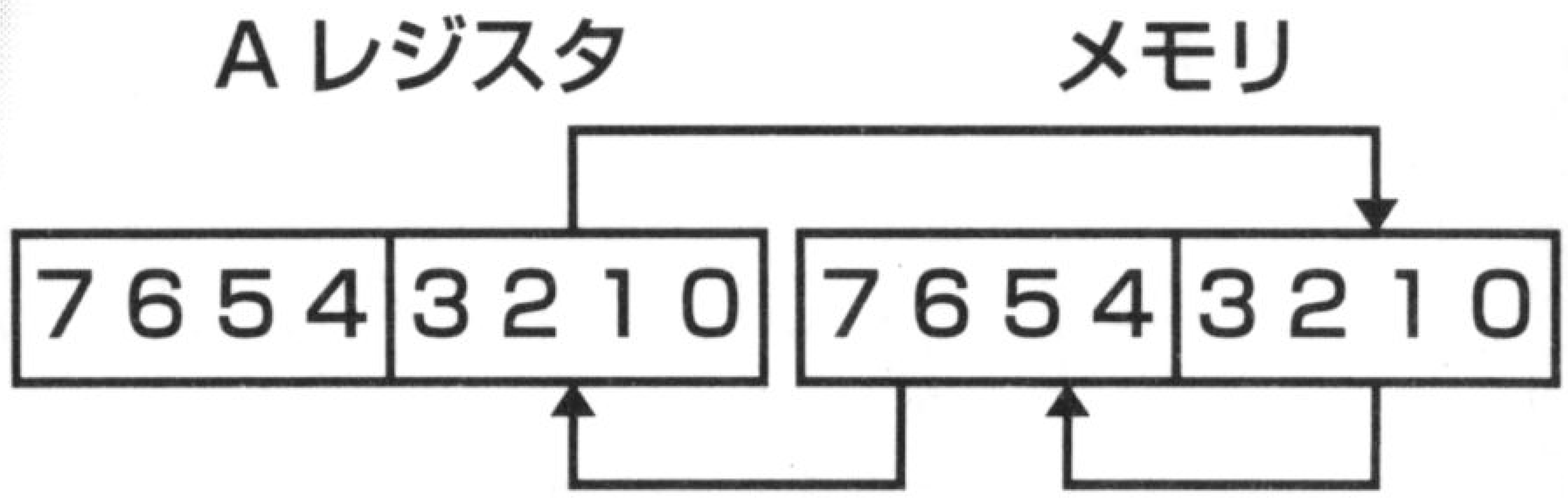
図7-86　RLA命令とRRA命令

rotate digit left and right between accumulator and location (HL)

RLD

ED 6F

HLレジスタペアが示すアドレスのメモリの内容と、Aレジスタの値の間で、次のような移動を行う。



S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	◆

パリティフラグ

この命令の動作を文章でまとめると、以下のようになります。

①メモリの下位４ビットを、メモリの上位４ビットに移動する。

②メモリの上位４ビットを、Aレジスタの下位４ビットに移動する。

③Aレジスタの下位４ビットを、メモリの下位４ビットに移動する。

④Aレジスタの上位４ビットは移動しない。

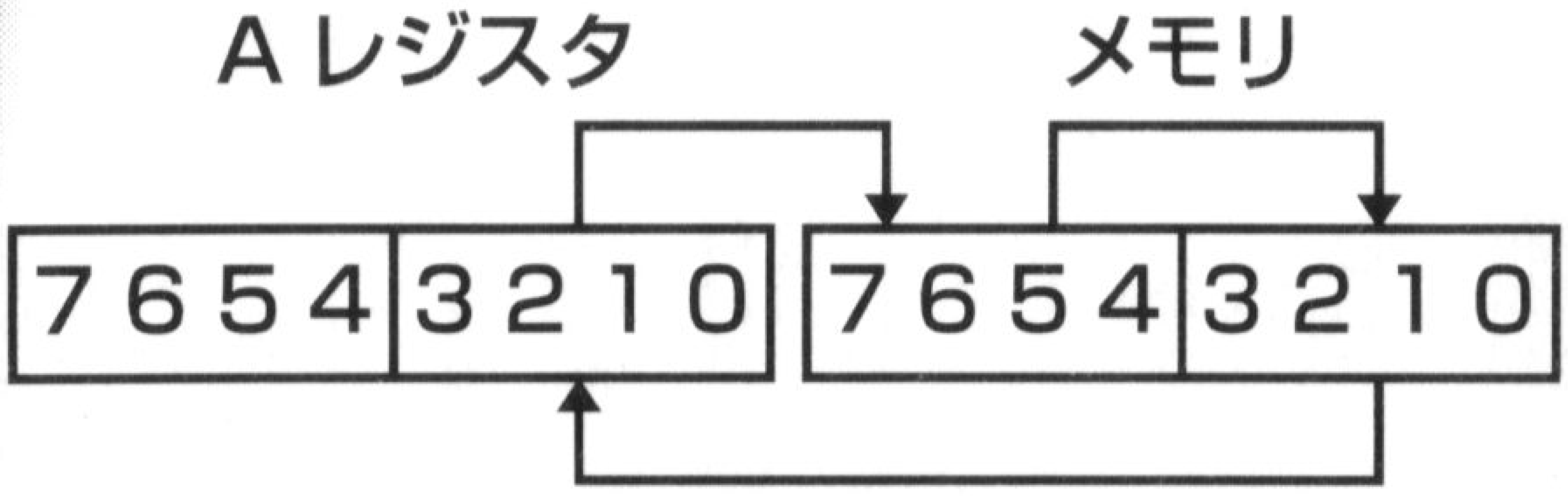
図7-87　RLD命令

rotate digit right and left between accumulator and location (HL)

RRD

ED 67

HLレジスタペアが示すアドレスのメモリの内容と、Aレジスタの値の間で、次のような移動を行う。



S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	◆

パリティフラグ

この命令の動作を文章でまとめると、以下のようになります。

①メモリの下位4ビットを、Aレジスタの下位4ビットに移動する。

②Aレジスタの下位4ビットを、メモリの上位4ビットに移動する。

③メモリの上位4ビットを、メモリの下位4ビットに移動する。

④Aレジスタの上位4ビットは移動しない。

図7-88　RRD命令

SLA (HL)　CB 26

HLレジスタペアが示すアドレスのメモリの内容を 1 ビットだけ左にシフトする。ビット 7 はキャリーフラグに、ビット 0 には 0 が入る。

CY	←	7	6	5	4	3	2	1	0	← 0

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

shift location (HL) left arithmetic

パリティフラグ

shift register r left arithmetic

SLA r

r レジスタの内容を 1 ビットだけ左にシフトする。ビット 7 はキャリーフラグに、ビット 0 には 0 が入る。

CY	←	7	6	5	4	3	2	1	0	← 0

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25

パリティフラグ

図7-89　SLA命令

SRA (HL)

HL レジスタペアが示すアドレスのメモリの内容を 1 ビットだけ右にシフトする。ビット 0 はキャリーフラグに、ビット 7 は変化しない。

7 6 5 4 3 2 1 0 → CY

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

CB 2E

shift location (HL) right arithmetic

パリティフラグ

shift register r right arithmetic

SRA r

r レジスタの内容を 1 ビットだけ右にシフトする。ビット 0 はキャリーフラグに、ビット 7 は変化しない。

7 6 5 4 3 2 1 0 → CY

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D

パリティフラグ

図7-90　SRA命令

SRL (HL)　　CB 3E

HL レジスタペアが示すアドレスのメモリの内容を 1 ビットだけ右にシフトする。ビット 0 はキャリーフラグに、ビット 7 には 0 が入る。

0 → [7|6|5|4|3|2|1|0] → [CY]

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

shift location (HL) right logical

パリティフラグ

shift register r right logical

SRL r

r レジスタの内容を 1 ビットだけ右にシフトする。ビット 0 はキャリーフラグに、ビット 7 には 0 が入る。

0 → [7|6|5|4|3|2|1|0] → [CY]

S	Z	H	P/V	N	CY
↑↓	↑↓	0	↑↓	0	↑↓

r→	A	B	C	D	E	H	L
機械語	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D

パリティフラグ

図7-91　SRL命令

ここで、ローテート命令を用いた実験を1つ取り上げます。機械語は、「21 10 00 3E 01 32 10 00 CB 06 C3 08」です。このプログラムでは、メモリの0010H番地に1バイト（8ビット）を確保して、この値をローテートしていきます。

ただし、実験を進める上で大きな問題があります。どうやってビットの値がローテートする様子を観察するのか、という点です。

もちろん、16進数ですから数値を見ながら、頭の中で思い浮かべることも1つの方法ですが、できれば2進数で眺めたいものです。

これに適しているのが、ダンプウィンドウです。このウィンドウにはオプション機能があって、2進数表示に切り替えることができます。

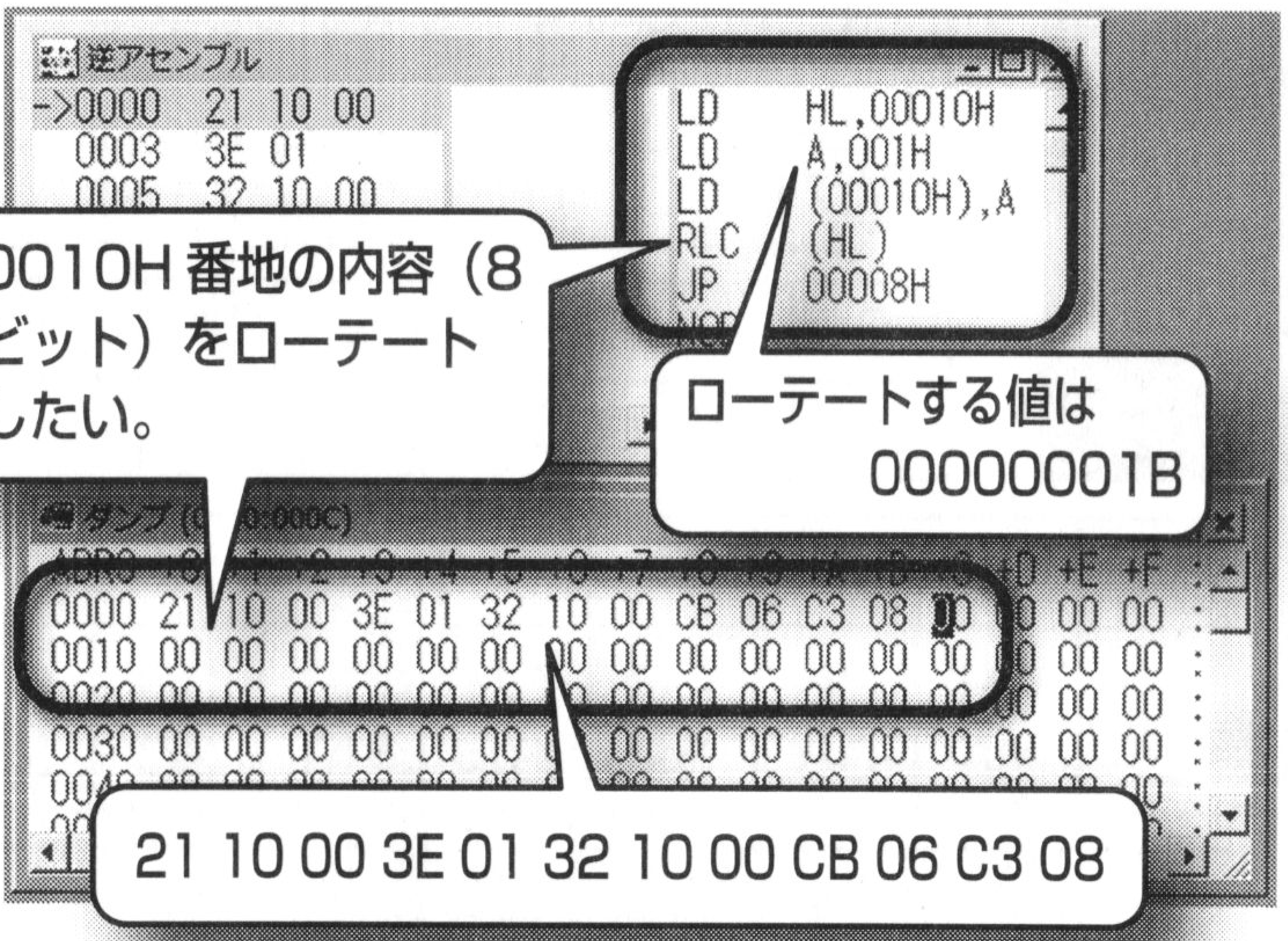


図7-92　ローテートの実験

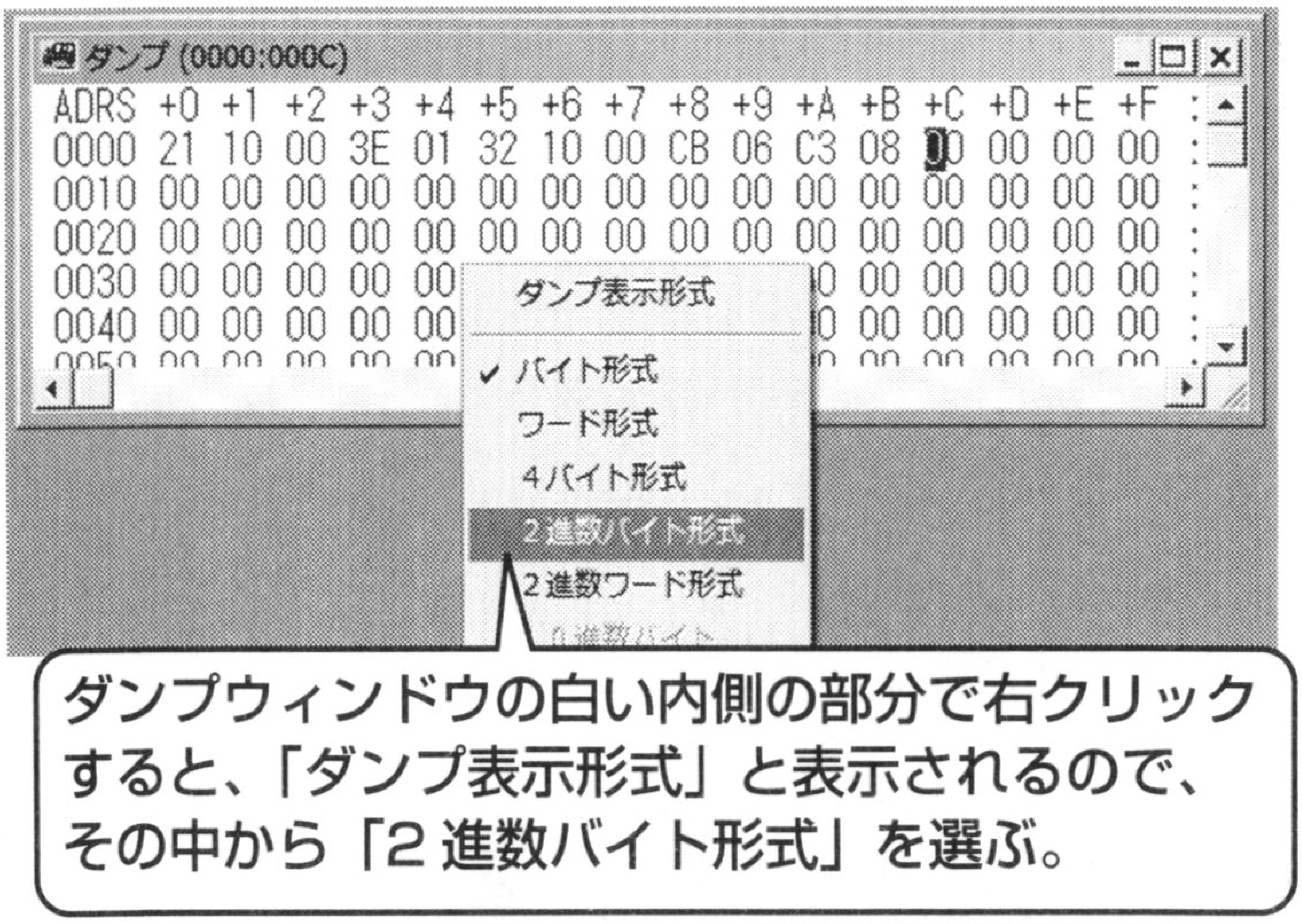


図7-93　ダンプウィンドウの２進数表示

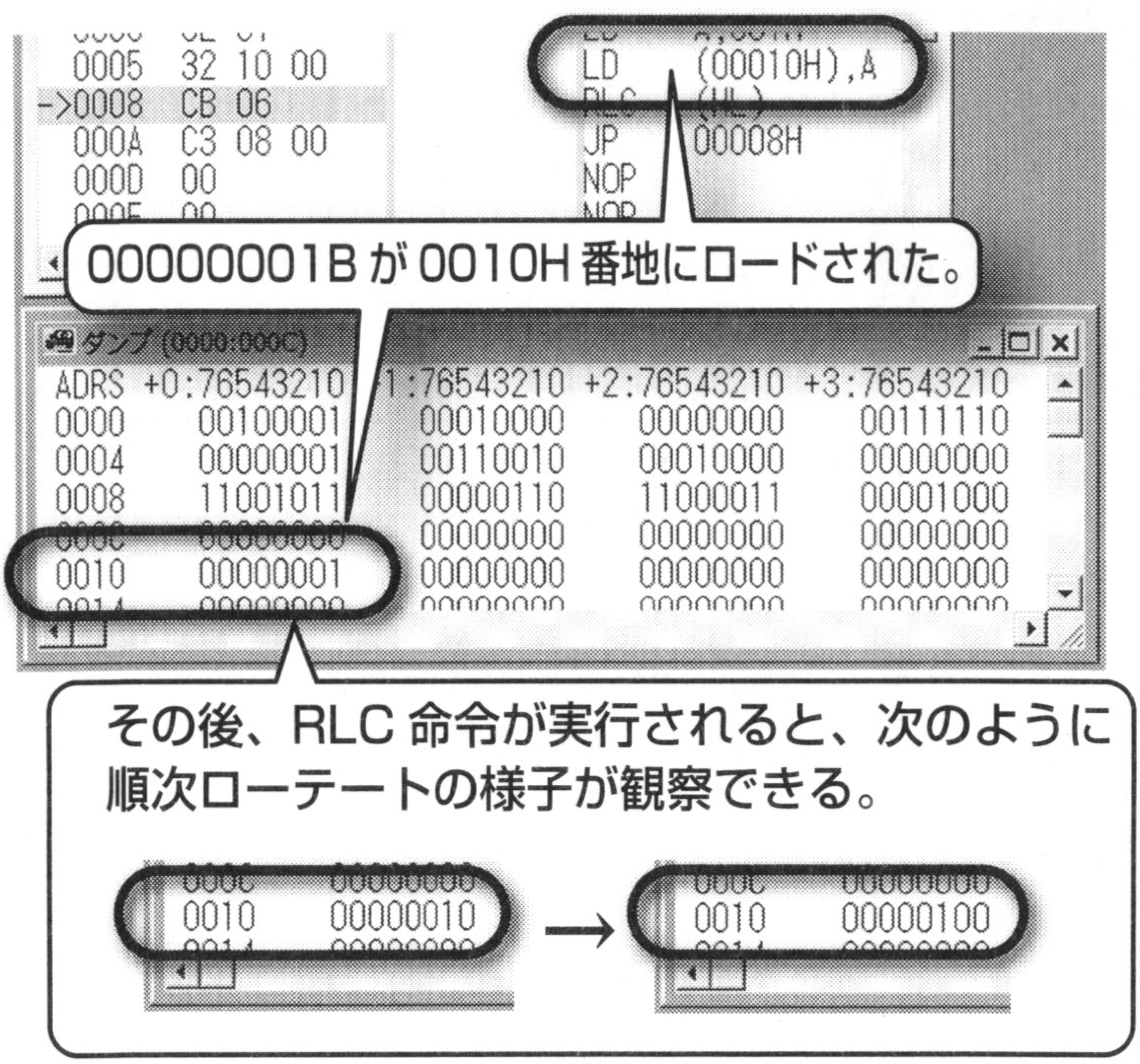


図7-94　２進数で実験する

7.13　ビット操作命令

ビット操作には、テスト、セット、リセットの３種類があります。

指定したレジスタの各１ビットを区別しながら、それらが「１」なのか「０」なのかをテストするのがビットテスト（BIT）命令です。また、各ビットを「１」にセットするのがセット（SET）命令、「０」にリセットするのがリセット（RES）命令です。

the bit of location (HL)

BIT b,(HL)

HL レジスタペアが示すアドレスの内容について、指定したビット b の値をチェックする。0 ならば Z フラグが 1 になる。

S	Z	H	P/V	N	CY
×	↑↓	1	×	0	◆

b→	7	6	5	4	3	2	1	0
機械語	CB 7E	CB 76	CB 6E	CB 66	CB 5E	CB 56	CB 4E	CB 46

図7-95　BIT命令（1）

the bit of register r

BIT b,r

r レジスタの内容について、指定したビット b の値をチェックする。0 ならば Z フラグが 1 になる。

S	Z	H	P/V	N	CY
×	↑↓	1	×	0	◆

b \ r	A	B	C	D	E	H	L
0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45
1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D
2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55
3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D
4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65
5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D
6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75
7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D

図7-96　BIT命令（2）

SET b,(HL)

HLレジスタペアが示すアドレスの内容について、指定したビットbの値を1にする。 ◆フラグは変化しない。

b→	7	6	5	4	3	2	1	0
機械語	CB FE	CB F6	CB EE	CB E6	CB DE	CB D6	CB CE	CB C6

set bit b of location (HL)

set bit b of register r

SET b,r

rレジスタの内容について、指定したビットbの値を1にする。◆フラグは変化しない。

b \ r	A	B	C	D	E	H	L
0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5
1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD
2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5
3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD
4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5
5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED
6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5
7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD

図7-97　SET命令

RES b,(HL)

HLレジスタペアが示すアドレスの内容について、指定したビットbの値を0にする。　◆フラグは変化しない。

b→	7	6	5	4	3	2	1	0
機械語	CB BE	CB B6	CB AE	CB A6	CB 9E	CB 96	CB 8E	CB 86

reset bit b of location (HL)

reset bit b of register r

RES b,r

rレジスタの内容について、指定したビットbの値を0にする。◆フラグは変化しない。

b＼r	A	B	C	D	E	H	L
0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85
1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D
2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95
3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D
4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5
5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD
6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5
7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD

図7-98　RES命令

7.14　汎用AF操作命令

Aレジスタには、ほかのレジスタにはできない固有の機能がいくつもあります。

NEG命令は記憶している値の「+」と「−」を反転し、CPL命令は8ビットの「0」と「1」を反転します。また、DAA命令によって、2進化10進数が求められます。

一方、Aレジスタに付随するフラグのうち、最も身近なのがCY（キャリーフラグ）ですが、この値を直接操作することができます。SCF命令はCY＝1とし、CCF命令は「1」と「0」を反転させます。

negate accumulator

オーバーフローフラグ

命令	S	Z	H	P/V	N	CY	コード
NEG	Aレジスタの値の符号（正と負）を反転する。						ED 44
	↑↓	↑↓	↑↓	↑↓	1	↑↓	

2進数の世界での正と負の反転は、この場合、0からAレジスタの値を引くことを意味します。
たとえば、55H（01010101B、85D）にNEGを実行すると、次の引き算を実行することと同じ意味になります。

```
     100000000B（100H）
  −   01010101B（ 55H、  85D）
  ─────────────
      10101011B（ ABH、−85D）
```

図7-99　NEG命令

CPL

Aレジスタの各ビットの値を反転する。つまり、1は0に、0は1に変わる。

2F

S	Z	H	P/V	N	CY
◆	◆	1	◆	1	◆

complement accumulator

DAA

各種演算（ADD,ADC,INC,SUB,SBC,DEC,NEG）の後にDAAを実行すると、実行直前のフラグレジスタの状態（各フラグの0と1の組み合わせ）に応じてAレジスタの値を10進補正する。

27

S	Z	H	P/V	N	CY
↑↓	↑↓	↑↓	↑↓	◆	↑↓

decimal adjust accumulator

パリティフラグ

10進補正とは、2進化10進数(BCD)に変換することです。たとえば59H+38Hの加算の結果がAレジスタに入ると、そのままでは16進数で「91H」となりますが、その直後にDAAを実行すると「97H」つまりHをはずせば、10進数で59+38=97が実行されたように補正してくれるのです。
なお、DAAの実行後のフラグの状態は、前に行った演算の種類やAレジスタの値によって異なります。

図7-100　CPL命令とDAA命令

SCF	キャリーフラグを１にする。						37
	S	Z	H	P/V	N	CY	
	◆	◆	0	◆	0	1	

set carry flag

CCF	キャリーフラグが反転し、Hフラグには、命令実行前のキャリーフラグの値が入る。						3F
	S	Z	H	P/V	N	CY	
	◆	◆	↑↓	◆	0	↑↓	

complement carry flag

図7-101　SCF命令とCCF命令

以上で、Z-Visionを用いた実験用命令語の解説を終えます。

掲載したサンプルプログラムで取り上げた命令語はZ80全体のごく一部ですが、いずれも基本的で使用頻度の高いものです。読者の皆さんは興味を持った命令語を１つ見つけたら、ぜひそれを使ってオリジナルの実験プログラムを作ってみてください。そして、「１行トレース」ボタンを押す直前に、その都度どのウィンドウでどのような変化が

起きるのかを予測してみるのです。そうすれば、やがて皆さんはZ-Visionなしでも、CPUの動きを頭の中でシミュレートできるようになるでしょう。

参考図書

①額田忠之『Z80ファミリ・ハンドブック』(CQ出版社、1985年1月)

②横山直隆『ロボット製作を目的としたZ80マイコン短期マスタ(増補版)』(シータスク、2004年3月)

③西久保靖彦『図解入門　よくわかるCPUの基本と仕組み』(秀和システム、2004年9月)

④伊東規之『マイクロコンピュータの基礎』(日本理工出版会、1990年10月)

⑤渡波　郁『CPUの創りかた』(毎日コミュニケーションズ、2003年9月)

⑥青木　勲『Z-Vision Plus Ver3.0総合マニュアル』(システムロード、1990年10月)

⑦青木　勲『XA80 Ver2.1ユーザーズマニュアル』(システムロード、1990年10月)

索引

【記号・数字】

【アルファベット】

【あ行】

【か行】

【ま行】

【ら行】

【Z-Visionのウィンドウ、ボタンなど】

N.D.C.548.22　219p　18cm

ブルーバックス　B-1665

動かしながら理解するCPUの仕組み
CD-ROM付

パソコンの中心はどうなっているのか

2010年1月20日　第1刷発行

著者　加藤ただし
発行者　鈴木　哲
発行所　株式会社講談社
〒112-8001 東京都文京区音羽2-12-21
電話　出版部　03-5395-3524
販売部　03-5395-5817
業務部　03-5395-3615
印刷所　(本文印刷) 豊国印刷株式会社
(カバー表紙印刷) 信毎書籍印刷株式会社
本文データ制作　講談社プリプレス管理部
製本所　株式会社国宝社

ISBN978-4-06-257665-9

発刊のことば

科学をあなたのポケットに

二十世紀最大の特色は、それが科学時代であるということです。科学は日に日に進歩を続け、止まるところを知りません。ひと昔前の夢物語もどんどん現実化しており、今やわれわれの生活のすべてが、科学によってゆり動かされているといっても過言ではないでしょう。

そのような背景を考えれば、学者や学生はもちろん、産業人も、セールスマンも、ジャーナリストも、家庭の主婦も、みんなが科学を知らなければ、時代の流れに逆らうことになるでしょう。

ブルーバックス発刊の意義と必然性はそこにあります。このシリーズは、読む人に科学的に物を考える習慣と、科学的に物を見る目を養っていただくことを最大の目標にしています。そのためには、単に原理や法則の解説に終始するのではなくて、政治や経済など、社会科学や人文科学にも関連させて、広い視野から問題を追究していきます。科学はむずかしいという先入観を改める表現と構成、それも類書にないブルーバックスの特色であると信じます。

一九六三年九月

野間省一

さらに知識の幅を広げるために、次のようなブルーバックスがあります。

『図解・わかる電子回路』（加藤 肇／見城尚志／高橋 久 B1084）電子科学生のベストセラー

『CD-ROM付 電子回路シミュレータ入門 増補版』（加藤ただし B1489）電子回路シミュレータを使ってみよう

『図解 つくる電子回路』（加藤ただし B1553）正しい工具の使い方、うまく作るコツ

『手作りラジオ工作入門』（西田和明 B1573）聴こえたときの感動がよみがえる

●Bの数字はブルーバックスの刊行順番号です。

ISBN978-4-06-257665-9

C0254 ¥1200E (0)

9784062576659

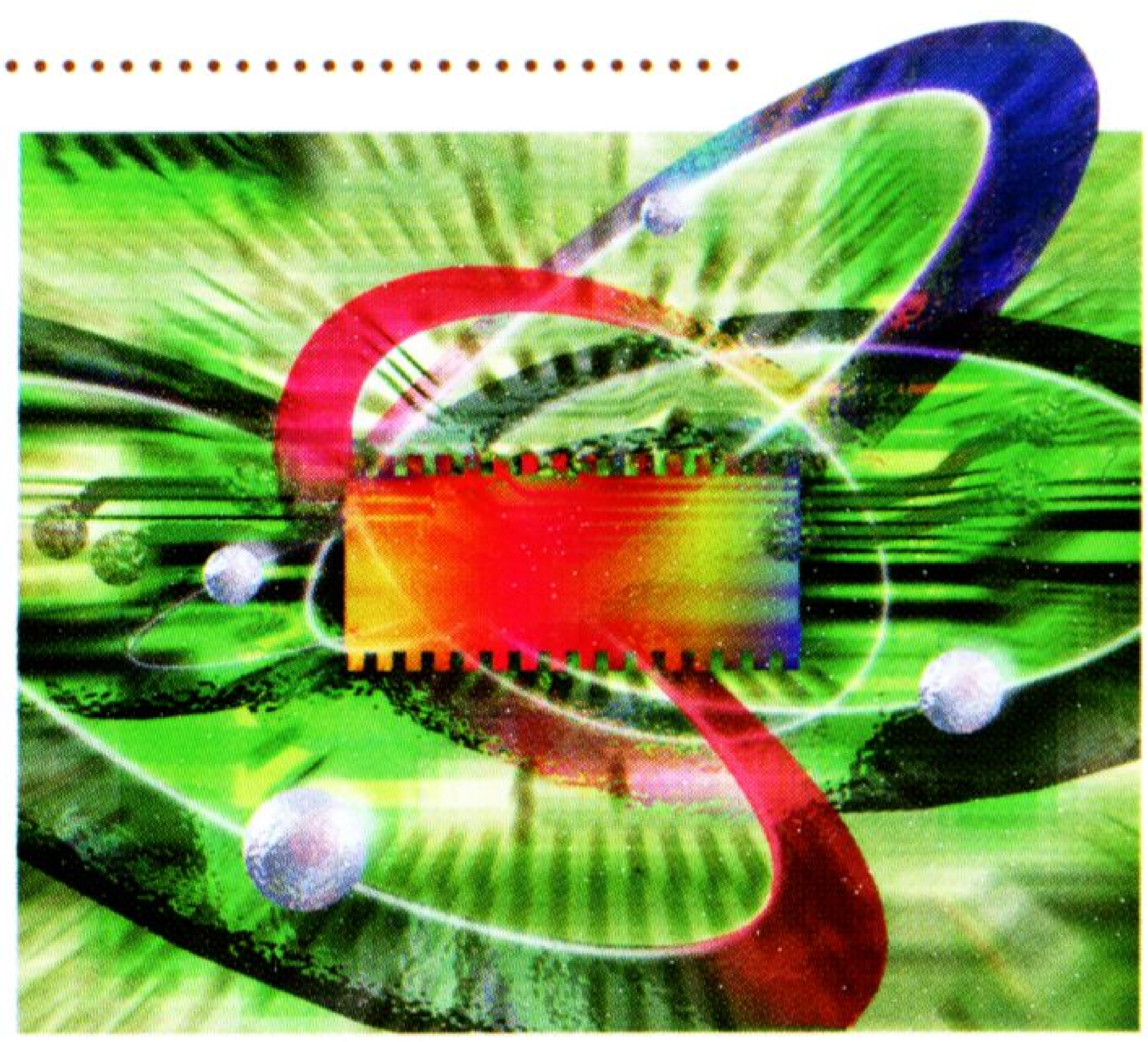
価格:本体 1200円(税別)

1920254012002

シミュレータで「Z80」を動かしてみよう

CPUの仕組みと働きを理解するには、
実際に動かしてみることが一番です。
本書付録のシミュレータソフトで
古典的CPU「Z80」に直接命令してみよう。

CPUでは実際にどんなことが行われているか
Windows画面で確認ができるので、
楽しみながらCPUの仕組みを理解できます。




動かしながら理解するCPUの仕組み
加藤ただし 著
「Z-Vision付録版」
収録
CD-ROM
COMPACT
disc
for Windows
BLUE BACKS
講談社 ブルーバックス B1665